uc3m | Universidad Carlos III de Madrid

Grado en Ingeniería Informática

2017-2018

Trabajo Fin de Grado

"Asesor automático para programadores noveles en lenguaje Java"

Víctor Murillo Fernández

Tutor/es

Valentín Moreno Pelayo

Lugar y fecha de presentación

En Universidad Carlos III de Madrid, Leganés a 11-07-2018

RESUMEN

El presente documento, expone el trabajo fin de grado del alumno Víctor Murillo Fernández, del grado de Ingeniería Informática de la Universidad Carlos III de Madrid. El proyecto se presenta bajo el nombre de “Asesor automático para programadores noveles en lenguaje Java”.

Se toma la decisión de realizar una aplicación web en lugar de una aplicación de escritorio, evitando así la instalación en un equipo y pudiendo ser accesible desde cualquier dispositivo que esté conectado a la red.

EL objetivo principal de la herramienta es la detección de errores sobre programas implementados en Java, y el posterior aviso al usuario de una manera clara y especifica del tipo de error detectado y su localización dentro del documento, además incluye un apartado donde se pueden visualizar gráficos con la cantidad de errores de todos los usuarios que utilizan la aplicación, pudiendo ser de ayuda para un posterior análisis. Adicionalmente, la aplicación detecta los comentarios del código analizado y permite al usuario su descarga para facilitar la documentación del programa, si fuese necesaria.

Con el uso de la herramienta, se pretende que el usuario sea capaz de solucionar sus propios errores y aprender de ellos, permitiendo a este lograr sus objetivos de programar en Java, creando códigos sin errores, correctamente estructurados y documentados.

La aplicación ha sido realizada en su totalidad por el alumno, siempre asesorado y apoyado por el tutor del proyecto Valentín Moreno Pelayo.

Palabras claves:

Proyecto Software, Aplicación Web, Análisis Código, Java, Errores.

DEDICATORIA

Me gustaría dedicar unas palabras y agradecer a todas esas personas que me apoyaron a lo largo de este camino en la Universidad.

En primer lugar, a mi familia, por darme la oportunidad de cumplir un sueño, en especial a mis padres, que no han dejado de apoyarme ni un solo segundo sin dejarme tirar la toalla en aquellos momentos duros al comienzo de la aventura, además de realizar un sobreesfuerzo para que pueda llegar donde estoy. Sin olvidarme de mi hermana, que me ayudó a la hora de motivarme y me facilito mucho la vida en estos últimos dos años en la universidad. A mis abuelos, por motivarme con aquella frase que se ha vuelto tan popular durante los últimos años en España "Estudia hijo, que la vida está muy mal".

En segundo lugar, a mis amigos, tanto a los de Leganés como a los del pueblo. Puedo decir que llegue a Leganés solamente con una familia postiza pero hoy puedo decir orgulloso que ya tengo dos. Quiero agradecer a los de Leganés por crear esta nueva familia en la que apoyándonos unos a otros hemos conseguido nuestros objetivos, y a los amigos del pueblo, que me ayudarán a desconectar completamente de mi vida, lo que me permitía regresar de nuevo con las pilas cargadas.

Una mención especial para aquella persona que supuso en punto de inflexión en la carrera, mi amigo, mi hermano A. Martín. Juntos cambiamos el rumbo de nuestras carreras, juntos superemos las asignaturas a las que nos enfrentábamos y juntos hemos logrado llegar a la meta.

No puedo olvidarme de aquellos profesores, que consiguieron motivarme, despertando en mí las ganas de aprender y la curiosidad del porqué de las cosas que aún conservo.

Con este proyecto cierro una de las etapas más duras y bonitas de mi vida, pero no va a ser la última y me gustaría seguir contando con todos, muchas gracias.

"Alea iacta est"

Julio César

ÍNDICE

.

ÍNDICE TABLAS

ÍNDICE ILUSTRACIONES

INDICE DE ECUACIONES

1. INTRODUCCION

1.1. Motivación del trabajo

Sin la necesidad de tener que acudir a un estudio estadístico, puedo asegurar de primera mano que el primer año que nos enfrentamos a una asignatura de programación, sin tener ningún conocimiento previo, supone un reto muy difícil. Esto deriva en un índice de suspensos mayor del que nos gustaría en la asignatura de programación 1 en el primer año del grado de Ingeniería Informática, lo que aumenta aún más los suspensos o abandonos en la segunda asignatura de programación cursada en el segundo cuatrimestre, que está estrictamente relacionada con la primera.

Por otra parte, en la actualidad, vivimos en una época digital en la que estamos rodeados de nuevas tecnologías (internet, ordenadores, móviles…) las cuales están basados en la programación. Esto implica que surja una nueva necesidad en el mundo laboral y a su vez nuevos puestos de trabajos [1], lo que conlleva que saber programar puede ayudar a abrir puertas a la hora de incorporarse al mundo laboral.

Para reducir la curva de aprendizaje que supone la primera vez que te enfrentas a un lenguaje de programación, se ha decidido crear esta herramienta que permita al usuario aprender de forma independiente y corregir sus propios errores, permitiendo a este mejorar a la hora de programar y conseguir así reducir el alto índice de suspenso en asignaturas relacionadas con la programación y evitar el desapego por esta asignatura.

Con el desarrollo de este proyecto no solamente se ayudará a usuarios que están aprendiendo a programar en el lenguaje Java si no que me servirá como carta de presentación en un futuro, me permitirá afianzar y poner a prueba todos mis conocimientos aprendidos a lo largo de la carrera y seguramente aprender alguna nueva tecnología.

1.2. Objetivos

El objetivo principal de este proyecto es ayudar reducir el número de errores en los códigos de programación en lenguaje Java consiguiendo con esto unos códigos con mayor calidad, escalables y reutilizables [2].

Para conseguir el objetivo principal segmentaremos este en varios objetivos parciales:

- Ayudar a mejorar la legibilidad en un código de programación.
- Ayudar a mejorar la estructura en los programas.
- Ayudar a hacer un uso eficiente de las variables del código.
- Ayudar a documentar los códigos desarrollados.

Como objetivo secundario se pretende que los alumnos de cualquier universidad que cursen asignaturas de este estilo reduzcan el rechazo por esta clase de asignaturas y se consiga aumentar el índice de aprobados.

1.3. Marco Regulador

En este apartado se describe el marco regulatorio en el que se ve envuelto esta aplicación. Por un lado, para tratar los datos personales que maneja la aplicación hay que tener en cuenta la ley orgánica de protección de datos que afecta a nuestro país y por otro lado, a la hora del desarrollo de la aplicación hay que tener en cuenta el estándar de calidad del producto software ISO-2500.

La nueva ley de protección de datos que ha entrado en vigor el 25 de mayo de 2018 es una ley mucho más exigente que permite preservar de una manera más seguro los datos personales de los usuarios y decidir, de manera clara y sencilla, si dan el consentimiento o no a una aplicación para tener acceso a sus datos personales [3]. A la hora de desarrollar la aplicación, hay que tener en cuenta esta nueva normativa que supondrá mayor atención a la hora de tratar los datos de los usuarios para evitar cualquier conflicto legal.

Para desarrollar un software de calidad se seguirá el estándar ISO/IEC 25010 [4] que consta de las siguientes características:

- Compatibilidad: capacidad de que dos o más componentes de un mismo software para compartir información y/o interactuar para llegar a un fin.
- Usabilidad: capacidad del software para ser aprendido y usado.
- Fiabilidad: capacidad de un software para realizar sus funciones.
- Seguridad: capacidad para proteger los datos internos del software.
- Mantenibilidad: capacidad del software para ser actualizado de manera rápida y efectiva.
- Portabilidad: capacidad del software para ser ejecutado en distintos dispositivos y distintos entornos.

1.4. Estructura de la memoria

En este apartado se describe la estructura del contenido del proyecto, este se encuentra divido en 8 capítulos que se describen a continuación:

Capítulo 1 Introducción: en esta sección se explica la motivación por la que se ha decidido llevar a cabo este proyecto, los objetivos que se pretende conseguir una vez concluido, el marco regulador al que está sometido y la estructura de este.

Capítulo 2 Estado del arte: en esta sección se explica la situación actual a la que se enfrenta el proyecto, el marco donde se va a mover el proyecto y las características con las que contara la aplicación.

Capítulo 3 Análisis del sistema: en este apartado se describe el alcance del sistema, los casos de uso y los requisitos funcionales y no funcionales que se deben cumplir.

Capítulo 4 Diseño del sistema: en este apartado se describe la arquitectura a seguir y las herramientas utilizadas, tanto a nivel software como hardware, para realizar el proyecto.

Capítulo 5 Implementación del sistema: en este apartado se describe como se lleva a cabo la implementación, los lenguajes utilizados y la localización de las distintas partes de la arquitectura en el proyecto.

Capítulo 6 Pruebas: en este apartado se describen las distintas pruebas realizadas al sistema para comprobar el correcto funcionamiento de este.

Capítulo 7 Estimación y costes: en esta sección se describen las tareas realizadas a lo largo del proyecto, se incluye un diagrama de Gantt explicativo y los costes que supone un proyecto estas características.

Capítulo 8 Conclusiones: en esta sección se presentan las conclusiones obtenidas una vez realizado el proyecto y si se han llegado a completar los objetivos marcados al principio de este.

2. ESTADO DEL ARTE

2.1. Situación actual.

Un estudio realizado por la universidad de Kent analizo millones de códigos realizados por estudiantes noveles que cursaban la asignatura de programación para el lenguaje Java. De este análisis de dedujo cuales eran los errores más comunes que cometían los estudiantes a la hora de programar.

Los errores más comunes cometidos por estos usuarios los podemos dividir en dos grandes grupos, los de sintaxis y los de lógica. Lo errores de sintaxis llevaban a la no compilación del código. Los errores de lógica permitían la correcta compilación del código, pero producían resultados no esperados. En este estudio se especifican cuáles son los 10 errores más frecuentes [5].

A continuación, se listan un resumen de estos errores:

- Uso incorrecto de paréntesis, llaves, corchetes o comillas.
- Llamadas a métodos con el numero incorrecto de argumentos.
- No retornar valor en funciones.
- Confundir el operador de asignación = con el de comparación ==.

Una vez que se llega a identificar y controlar los errores, se producen códigos de mayor eficacia y eficiencia, aunque la única manera de llegar a esta situación es la experiencia y la comprensión de los errores.

Haciendo referencia a un aspecto de las motivaciones del proyecto, un usuario que sepa evitar los errores puede extrapolar ese conocimiento a otros lenguajes o profundizar más en el mismo, lo que permitirá superar asignaturas de más nivel y convertirse en usuarios expertos.

2.2. Marco actual.

Como se comenta en el apartado anterior, a la hora de aprender a programar se cometen multitud de errores de todo tipo, esto no quiere decir que no se cometan errores una vez que ya tienes experiencia.

Los errores a la hora de programar se pueden clasificar en dos grandes conjuntos, los de compilación y los de ejecución [6].

Los errores de compilación son los que se producen cuando el compilador realiza una "lectura previa del código" y comprueba si la estructura es correcta, los errores de este tipo provocan que el código no llegue a ejecutarse. Este tipo de errores podemos dividirlo en los siguientes subtipos:

- Errores de sintaxis: estos errores se producen al escribir de forma incorrecta alguna sección del código, por ejemplo, no terminar las instrucciones con el carácter punto y coma [7].
- Errores de procesos no validos en compilación: estos errores se producen cuando se hace un llamamiento a algún elemento que no existe o que tiene una estructura diferente, por ejemplo, utilizar una variable no declarada [8].

Lo errores de ejecución son los que el compilador no es capaz de detectar, y permite que el programa se ejecute con normalidad hasta llegar a la instrucción que es incorrecta y no permite al programa que finalice con normalidad. Estos errores se pueden dividir en los siguientes subtipos:

- Errores de procesos no validos en ejecución: estos errores se producen cuando la estructura de la instrucción está bien diseñada, pero se quiere realizar algo que no es posible, por ejemplo, la división por cero.
- Errores lógicos de bucle infinito: estos errores se producen cuando se declara un bucle y no se gestionan de forma correcta sus parámetros de control.
- Lógicos tipo resultado incorrecto: estos errores están relacionados con el diseño que ha tomado el programador, es un error que el programa no puede detectar por sí mismo, sino que es el propio programador el que debe revisar el código y comprobar el correcto funcionamiento del flujo de programa y sus operaciones.

Una vez que se conocen los errores que se pueden cometer a la hora de programar, tenemos que hablar de los entornos de programación donde llevar a cabo la implementación de dichos programas.

Existen muchos entornos de programación donde se podría desarrollar un programa en lenguaje Java como Netbeans, Eclipse, Intellij IDEA, … En este proyecto se ha optado por el entorno de Netbeans por lo que nos centraremos en él.

La herramienta de nos permite el desarrollo de programas tanto de escritorio como web, además de proporcionarnos un gran asistente que nos avisa de múltiples errores de compilación, también cuenta con una función de autocompletado que nos permite

añadir los métodos más comunes a clases (GET, SET,) y completar instrucciones de manera sencilla.

Bajo mi punto de vista, este entorno de desarrollo sería bastante recomendable para usuarios noveles que están aprendiendo a programar, ya que como decimos previene avisando al programador donde está cometiendo errores de compilación y le permitirán centrarse solamente en errores de ejecución y el correcto funcionamiento del programa.

2.3. Diseño de soluciones.

Como se menciona en el apartado anterior, los entornos de desarrollo cuentan con asistentes que le avisan al programador de los errores en tiempo de compilación, por eso se optará en la creación de una herramienta que ayude al programador a detectar los errores de ejecución y le ayude en la creación de programas legibles, modulares, con un diseño estético perfecto y ayude a la hora de documentar la funcionalidad del programa.

La herramienta contará con los siguientes elementos:

- Será una aplicación web para que pueda ser accesible desde cualquier dispositivo conectado a la red y poder realizar el análisis del código desde cualquier lugar.
- Gestión de usuarios para controlar el acceso a la aplicación y poder llevar un control de la cantidad de usuarios que la están utilizando.
- La herramienta proporcionara una lista de análisis a realizar sobre el código seleccionado proporcionando un feedback al usuario del tipo de error que se y donde se encuentra en el código.
- La herramienta permitirá al usuario descargarse un fichero con los comentarios detectados en el código que facilitará la documentación del código analizado.
- La herramienta proporcionará al usuario unas estadísticas globales con la cantidad de errores cometidos por todos los usuarios para tomar como referencia y hacer hincapié para intentar evitarlos.

Con el diseño de esta herramienta, se pretende que el usuario novel sea capaz de corregir por sí mismo sus propios fallos y al mismo tiempo aprender, para en un futuro poder evitarlos y cumplir todos los objetivos al comienzo del documento.

3. ANÁLISIS DEL SISTEMA

En este apartado se definen el alcance del proyecto los casos de uso, los requisitos del sistema y la matriz de trazabilidad que relaciona casos de uso con requisitos.

3.1. Alcance.

El sistema está pensado para el análisis de código de usuarios noveles que se encuentran dando sus primeros pasos en la programación Java. Con el uso del sistema este tipo de usuarios podrá localizar de manera sencilla los errores del código y poder alcanzar el objetivo de producir un código de calidad.

Esta herramienta también podrá ser utilizada por Profesores, facilitando así la corrección y evaluación del código de alumnos.

3.2. Casos de uso.

En este apartado se van a describir los casos de usos posibles de la aplicación. Utilizando la técnica de definición de casos de usos nos permitirá obtener con mayor facilidad los requisitos software de nuestra aplicación. [**9**]

Para la mejor comprensión de los casos de uso los representaremos mediante diagramas UML en los que tenemos que definir primero quienes son los actores (roles del usuario del sistema) y los casos de usos (operaciones que se desean que realice el sistema).

Los actores son representados mediante:

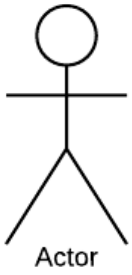

Fig. 1: Definición actor

Los casos de usos son representados mediante:

Fig. 2: Definición caso de uso

Como solamente tenemos un tipo de rol en la aplicación los actores se nombran como usuario.

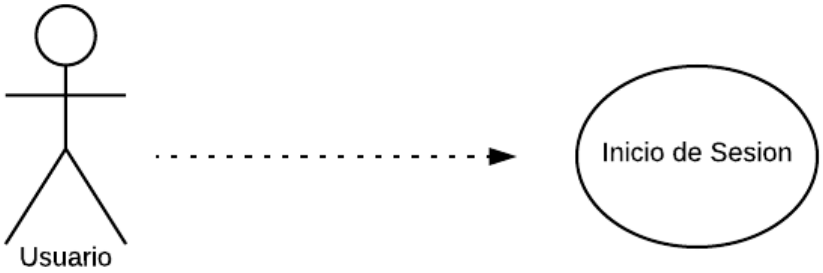

Fig. 3: Caso de uso 1

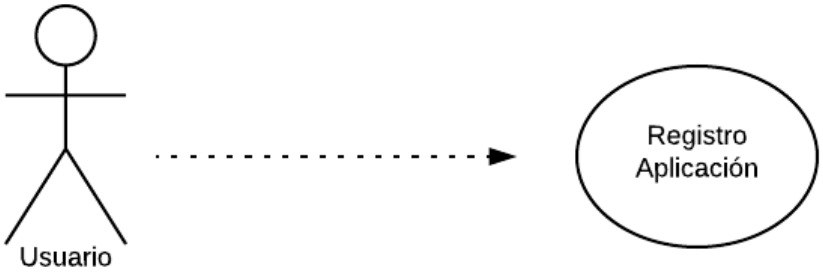

Fig. 4: Caso de uso 2

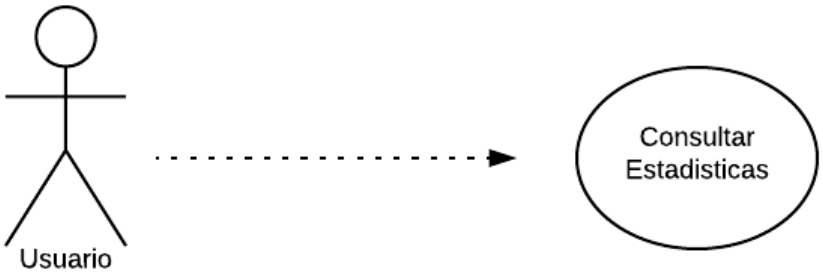

Fig. 5: Caso de uso 3

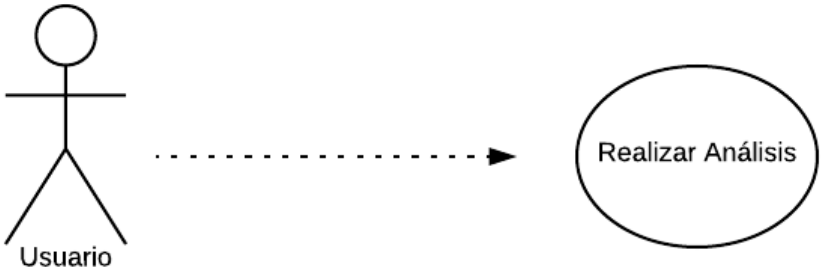

Fig. 6: Caso de uso 4

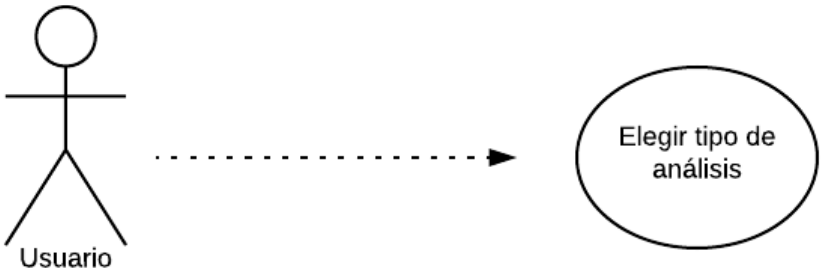

Fig. 7: Caso de uso 5

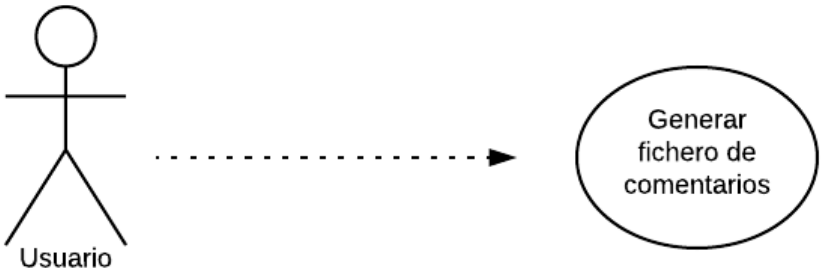

Fig. 8: Caso de uso 6

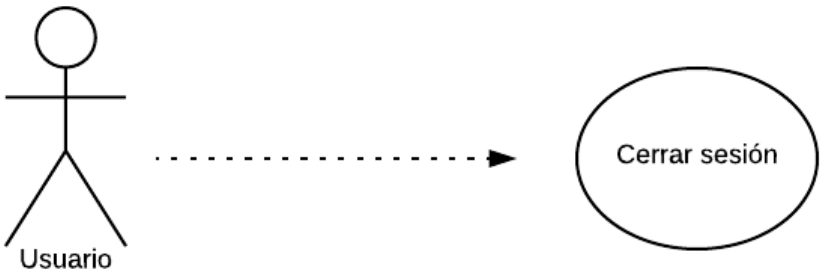

Fig. 9: Caso de uso 7

3.3. Requisitos.

En este apartado se van a describir los requisitos que tiene que cumplir el sistema. Clasificaremos los requisitos en dos tipos: funcionales y no funcionales. Los requisitos funcionales hacen referencia a la funcionalidad al sistema y los no funcionales a como se van a realizar las funcionalidades. [10].

Crearemos la siguiente tabla para la definición requisitos:

Tabla 1: Plantilla de requisitos

RX-Y	
Nombre	Nombre del requisito
Prioridad	[Alta, Media, Baja]
Coste	[Alto, Medio, Bajo]
Descripción	Breve descripción del requisito.

Los campos de la tabla serán los siguientes:

- Identificador: código alfanumérico univoco dado a cada requisito en el proyecto con el formato RX-Y, donde:
    - X: Toma los valores F o NF dependiendo si es un requisito funcional o no funcional respectivamente.
    - Y: Número natural que aumenta una unidad a medida que se añaden nuevos requisitos del mismo tipo. El primer requisito de cada tipo comienza con el número 1.
- Nombre: nombre en lenguaje formal del requisito.
- Prioridad: importancia de implementación del requisito en el proyecto. Toma los valores de Alta, Media y Baja.
- Coste: esfuerzo de implementación necesario para realizar el requisito. Toma los valores de Alto, Medio y Bajo.

- Descripción: breve descripción en lenguaje formal que explica la necesidad y/o la funcionalidad del requisito.

3.3.1. Requisitos funcionales.

En esta sección se muestran los requisitos funcionales del proyecto en el formato de tabla tomando como referencia la tabla de plantilla de requisitos.

Tabla 2: Requisito funcional 1

RF-1	
Nombre	Registro Usuario
Prioridad	Media
Coste	Medio
Descripción	La aplicación permite el registro de usuarios mediante la petición de un nombre, un email y una contraseña. El email no debe estar registrado con anterioridad.

Tabla 3: Requisito funcional 2

RF-2	
Nombre	Login usuario
Prioridad	Media
Coste	Medio
Descripción	La aplicación permite ingresar a un usuario mediante la petición de un email y una contraseña registrados con anterioridad.

Tabla 4: Requisito funcional 3

RF-3	
Nombre	Selección análisis de código
Prioridad	Baja
Coste	Bajo
Descripción	Permite el acceso al usuario a la pantalla de análisis de código.

Tabla 5: Requisito funcional 4

RF-4	
Nombre	Seleccionar estadísticas del sistema.
Prioridad	Baja
Coste	Medio
Descripción	Permite el acceso al usuario a la pantalla de gráficos estadísticos.

Tabla 6: Requisito funcional 5

RF-5	
Nombre	Mostrar gráficos estadísticos.
Prioridad	Baja
Coste	Medio
Descripción	La aplicación muestra gráficos de errores recogidos en la base de datos.

Tabla 7: Requisito funcional 6

RF-6	
Nombre	Selección de fichero a analizar.
Prioridad	Alta
Coste	Medio
Descripción	La aplicación permite al usuario seleccionar mediante una interfaz el archivo a analizar.

Tabla 8: Requisito funcional 7

RF-7	
Nombre	Fichero valido.
Prioridad	Baja
Coste	Medio
Descripción	La aplicación solo analiza ficheros con la extensión java.

Tabla 9: Requisito funcional 8

RF-8	
Nombre	Fichero invalido.
Prioridad	Baja
Coste	Bajo
Descripción	La aplicación avisa al usuario que el fichero seleccionado no contiene una extensión valida.

Tabla 10: Requisito funcional 9

RF-9	
Nombre	Selección de tipo de análisis.
Prioridad	Baja
Coste	Medio
Descripción	La aplicación permitirá al usuario seleccionar el tipo de análisis a realizar una vez seleccionado un tipo de fichero valido.

Tabla 11: Requisito funcional 10

RF-10	
Nombre	Múltiple selección.
Prioridad	Alta
Coste	Bajo
Descripción	La aplicación permite al usuario la selección de múltiples análisis.

Tabla 12: Requisito funcional 11

RF-11	
Nombre	Análisis mínimo.
Prioridad	Alta
Coste	Medio
Descripción	La aplicación no permite realizar un análisis si no se ha seleccionado al menos uno. La aplicación notifica de este error al usuario.

Tabla 13: Requisito funcional 12

RF-12	
Nombre	Análisis de legibilidad
Prioridad	Alta
Coste	Alto
Descripción	La aplicación realiza un análisis completo de la sección legibilidad (Análisis de indexación, análisis de formato de nombres, análisis de comentarios y análisis de saltos).

Tabla 14: Requisito funcional 13

RF-13	
Nombre	Análisis de modularidad.
Prioridad	Alta
Coste	Alto
Descripción	La aplicación realiza un análisis completo de la sección modularidad (Controlar tamaño de subprogramas, controlar valor de retorno de funciones y procedimientos, control de argumentos pasados por valor, control de impresiones de programas, control de datos de entrada por teclado, control de salida de datos por pantalla).

Tabla 15: Requisito funcional 14

RF-14	
Nombre	Análisis de uso de variables.
Prioridad	Alta
Coste	Alto
Descripción	La aplicación realiza un análisis completo de la sección de uso de variables (control de variable no inicializada, controlar la múltiple asignación de variables, control de variable no utilizada, control de acceso a variables globales, control de parámetros de bucles y control de variables de retorno en funciones).

Tabla 16: Requisito funcional 15

RF-15	
Nombre	Lectura fichero
Prioridad	Alta
Coste	Medio
Descripción	La aplicación realiza una lectura de un fichero para que posteriormente pueda ser analizado.

Tabla 17: Requisito funcional 16

RF-16	
Nombre	Análisis de sangrado de texto.
Prioridad	Alta
Coste	Medio
Descripción	La aplicación analiza el fichero para comprobar que todas las líneas están correctamente indexadas.

Tabla 18: Requisito funcional 17

RF-17	
Nombre	Análisis de formato de variables
Prioridad	Alta
Coste	Alto
Descripción	La aplicación analiza el fichero para comprobar que el nombre de las variables, las constantes, los procedimientos y las funciones siguen el mismo formato (CamelCase).

Tabla 19: Requisito funcional 18

RF-18	
Nombre	Análisis de comentarios
Prioridad	Alta
Coste	Medio
Descripción	La aplicación analiza el fichero para comprobar que todas las subrutinas están precedidas por un comentario.

Tabla 20: Requisito funcional 19

RF-19	
Nombre	Análisis de saltos incondicionales
Prioridad	Alta
Coste	Bajo
Descripción	La aplicación analiza el fichero para comprobar que no se realiza ningún salto incondicional.

Tabla 21: Requisito funcional 20

RF-20	
Nombre	Control de tamaño subprogramas.
Prioridad	Alta
Coste	Medio
Descripción	La aplicación analiza cada uno de los subprogramas del fichero y comprueba que no excedan el tamaño máximo.

Tabla 22: Requisito funcional 21

RF-21	
Nombre	Controlar valor de retorno de funciones.
Prioridad	Alta
Coste	Medio
Descripción	La aplicación analiza el código y controla que todas las funciones tengan un valor de retorno.

Tabla 23: Requisito funcional 22

RF-22	
Nombre	Control de valor de retorno de procedimientos.
Prioridad	Alta
Coste	Medio
Descripción	La aplicación analiza el programa y controla que los procedimientos no tengan valor de retorno.

Tabla 24: Requisito funcional 23

RF-23	
Nombre	Control de argumentos pasados por valor.
Prioridad	Alta
Coste	Alto
Descripción	La aplicación analiza el programa y controla que no se modifiquen los argumentos pasados por valor.

Tabla 25: Requisito funcional 24

RF-24	
Nombre	Control de impresiones de programas.
Prioridad	Alta
Coste	Alto
Descripción	La aplicación analiza el programa y controla que un método que contenga argumentos (para el cálculo de datos) no debe imprimir por pantalla.

Tabla 26: Requisito funcional 25

RF-25	
Nombre	Control de datos de entrada por teclado.
Prioridad	Alta
Coste	Medio
Descripción	La aplicación analiza el programa y controla que siempre que se pide un dato por pantalla debe haber un mensaje previo de petición.

Tabla 27: Requisito funcional 26

RF-26	
Nombre	Control de salida de datos por pantalla.
Prioridad	Alta
Coste	Medio
Descripción	La aplicación analiza el código y controla que siempre que se saque un dato por pantalla debe estar precedida de un rótulo.

Tabla 28: Requisito funcional 27

RF-27	
Nombre	Control de variables no inicializadas.
Prioridad	Alta
Coste	Medio
Descripción	La aplicación analiza el código y controla que no se utilice una variable que no ha sido inicializada con anterioridad.

Tabla 29: Requisito funcional 28

RF-28	
Nombre	Control de múltiple asignación a variable.
Prioridad	Alta
Coste	Alto
Descripción	La aplicación analiza el código y controla que no se le asigne un valor nuevo a una variable sin hacer uso del valor anterior.

Tabla 30: Requisito funcional 29

RF-29	
Nombre	Control de variable no utilizada.
Prioridad	Alta
Coste	Medio
Descripción	La aplicación analiza el programa y controla que las variables declaradas se utilizan mínimo una vez.

Tabla 31: Requisito funcional 30

RF-30	
Nombre	Control de acceso a variables globales.
Prioridad	Alta
Coste	Alto
Descripción	La aplicación analiza el fichero y controla que no se pueda acceder a las variables globales por visibilidad.

Tabla 32: Requisito funcional 31

RF-31	
Nombre	Control de parámetros de bucles.
Prioridad	Alta
Coste	Alto
Descripción	La aplicación analiza el fichero y controla que no se modifican los parámetros de control de bucles del tipo while y for.

Tabla 33: Requisito funcional 32

RF-32	
Nombre	Variable de retorno de función sin valor.
Prioridad	Alta
Coste	Alto
Descripción	La aplicación analiza el código y controla que el valor de retorno de una función contenga un valor.

Tabla 34: Requisito funcional 33

RF-33	
Nombre	Fichero de comentarios descargable.
Prioridad	Alta
Coste	Medio
Descripción	La aplicación analiza el código y recopila todos los comentarios para posteriormente generar un fichero que permita al usuario documentar con mayor facilidad el programa analizado.

3.3.2. Requisitos no funcionales.

En esta sección se muestran los requisitos no funcionales tomando como referencia la tabla de plantilla de requisitos.

Tabla 35: Requisito no funcional 1

RNF-1	
Nombre	Aplicación Web
Prioridad	Alta
Coste	Bajo
Descripción	La aplicación esta implementada para ser visualizada desde cualquier navegador web compatible con HTML5 y JavaScript.

Tabla 36: Requisito no funcional 2

RNF-2	
Nombre	Interfaz adaptativa.
Prioridad	Baja
Coste	Bajo
Descripción	La aplicación está dotada de una interfaz adaptativa para poder visualizarse en cualquier tipo de dispositivo.

Tabla 37: Requisito no funcional 3

RNF-3	
Nombre	Facilidad de uso de interfaz.
Prioridad	Alto
Coste	Bajo
Descripción	La aplicación cuenta con una interfaz fácil e intuitiva que permite al usuario navegar de manera sencilla.

Tabla 38: Requisito no funcional 4

RNF-4	
Nombre	Idioma interfaz
Prioridad	Alta
Coste	Bajo
Descripción	El idioma por defecto es español.

Tabla 39:: Requisito no funcional 5

RNF-5	
Nombre	Manual de usuario
Prioridad	Alta
Coste	Medio
Descripción	La aplicación cuenta con un manual de usuario para facilitar el aprendizaje de su uso.

3.4. Matriz de trazabilidad.

En este apartado se realiza una matriz de trazabilidad que relaciona los casos de uso con los requisitos funcionales para comprobar que se van a poder realizar todas las acciones propuestas e identificar los requisitos más críticos en función al número de casos de uso que afecten.

Tabla 40: Matriz de trazabilidad

	CU1	CU2	CU3	CU4	CU5	CU6	CU7
RF-1	x	x					x
RF-2	x						x
RF-3				x	x		
RF-4			x				
RF-5			x				
RF-6				x	x	x	
RF-7				x	x		
RF-8				x	x		
RF-9					x		
RF-10				x	x		
RF-11				x	x		
RF-12				x	x	x	
RF-13				x	x	x	
RF-14				x	x	x	
RF-15				x	x	x	
RF-16				x	x	x	

RF-17				x	x	x	
RF-18				x	x	x	
RF-19				x	x	x	
RF-20				x	x	x	
RF-21				x	x	x	
RF-22				x	x	x	
RF-23				x	x	x	
RF-24				x	x	x	
RF-25				x	x	x	
RF-26				x	x	x	
RF-27				x	x	x	
RF-28				x	x	x	
RF-29				x	x	x	
RF-30				x	x	x	
RF-31				x	x	x	
RF-32				x	x	x	

En la matriz se puede observar que los casos de uso 4,5 y 6 son los más críticos, en especial el número 5 que es el que está relacionado con toda la funcionalidad del sistema. Se puede deducir que la mayor parte del tiempo de proyecto estará dedicada a la implementación de estos casos de uso.

4. DISEÑO DEL SISTEMA.

En este apartado se especifica la arquitectura que tendrá el sistema, el entorno tecnológico donde se va a desarrollar y el diseño que se sigue para realizar cada una de las distintas capas de la arquitectura.

4.1. Arquitectura del sistema.

La arquitectura elegida para el sistema es modelo-vista-controlador (MVC). El MVC se caracteriza por la separación de la aplicación en tres capas diferentes para dividir la responsabilidad de la aplicación. [11]

- Modelo: es la capa que está relacionada con los datos, esta capa será la encargada de almacenar, eliminar o realizar modificaciones de los datos.
- Vista: es la capa relacionada con la interfaz, será la capa encargada de que el usuario interactúe con el sistema, realizando peticiones y visualizando los resultados.
- Controlador: es la capa que conecta la vista con el modelo, es la capa donde se realiza toda la lógica de la aplicación para poder dar sentido a las peticiones realizadas.

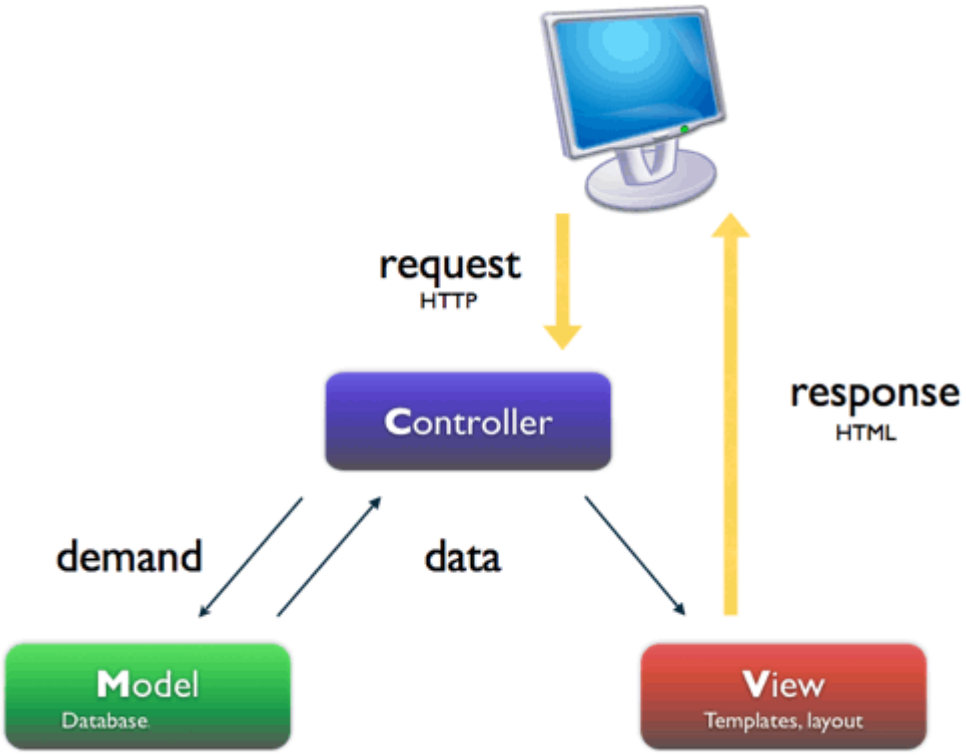

Fig. 10: Modelo vista controlador (MVC)

4.2. Entorno tecnológico.

En este apartado se van a describir las herramientas utilizadas para el desarrollo tanto a nivel de hardware como de software.

4.2.1. Hardware

A continuación, se describen las especificaciones de hardware con las que cuenta el equipo utilizado para el desarrollo:

- Procesador: Intel Core i5-7600K de 3.8GHz.
- Memoria RAM: 2 módulos de memoria DDR4 con una velocidad de 2400MHz con 8GB de capacidad y un tiempo de latencia de CL15
- Tarjeta gráfica: GeForce GTX 1060 con 6GB de capacidad.
- Disco duro: SATA3 de 1TB de capacidad.
- Monitor: Benq de 23.8" Full HD LED.

4.2.2. Software

Las herramientas software utilizadas en este proyecto son todas gratuitas por lo que no supone un coste adicional al presupuesto del proyecto.

NetBeans:

La mayor parte de la funcionalidad del sistema se ha desarrollado en lenguaje Java, para llevar a cabo esto se ha utilizado el entorno de desarrollo de NetBeans, que nos proporciona una fácil integración con las demás partes del sistema. Solamente se ha tenido que añadir la librería externa de MySQL Connectors para la comunicación con la base de datos.

Visual Studio Code

En el desarrollo de la parte visual se ha utilizado el entorno de Visual Studio Code que proporciona una funcionalidad de autocompletado de código para los lenguajes de HTML, CSS y JavaScript que nos permite avanzar de manera más rápida y eficiente en el desarrollo.

El lenguaje HTML es utilizado para la implementación de los elementos de la interfaz, con el lenguaje CSS le damos diseño a dichos elementos y con el lenguaje JavaScript podemos dotar al sitio de web de cierta funcionalidad para facilitar la experiencia del usuario.

Para el desarrollo de la interfaz nos hemos apoyado en dos librerías externas. La primera, Bootstrap, para mejor la visualización de los elementos del sistema, y la segunda Google Charts, para la implementación de gráficos de resultados.

MySQL Workbench

Los datos del sistema, tanto usuarios como resultados estadísticos, serán almacenados en una base de datos relacional, desarrollada en el lenguaje MySQL mediante la herramienta MySQL Workbench. Esta herramienta nos facilita el acceso a la creación, administración y consulta de datos mediante la interfaz.

Microsoft Word

Esta es la única herramienta de pago utilizada para la elaboración del proyecto, pero al disponer de licencias por parte de la universidad no se ha decidido incluir en el presupuesto. Esta herramienta ha sido utilizada para realizar la documentación del proyecto.

LucidChart

Herramienta online gratuita con la que se han realizado todos los diagramas de casos de uso que se incluyen en el presente documento.

Mockup

Herramienta gratuita de escritorio que ha sido utilizada para la creación de la maqueta de la interfaz del sistema que se tomará como referencia para la implementación final.

SmartSheet

Herramienta online gratuita durante los primeros treinta días, con esta herramienta se ha diseñado el diagrama de Gantt para la planificación del proyecto.

4.3. Diseño de la interfaz (Vista).

En este apartado se van a explicar los campos y las funcionalidades de cada una de las distintas vistas del sistema. Las interfaces que se muestran a continuación han sido realizadas mediante la herramienta de diseño de Balsamiq mockup.

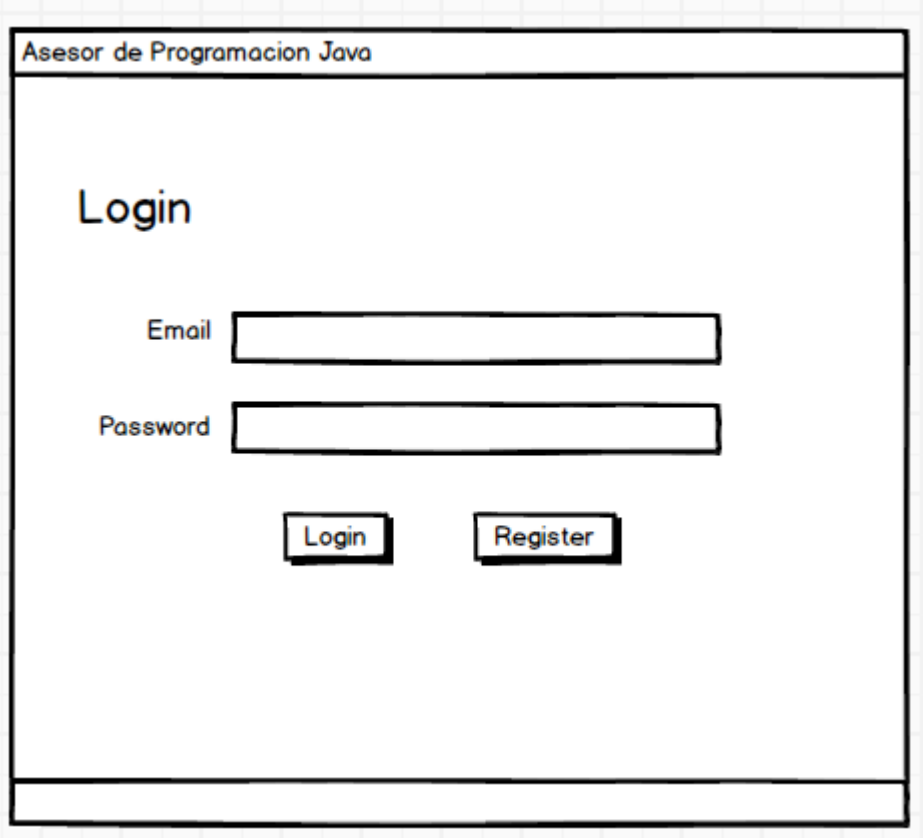

Fig. 11: Plantilla de Login

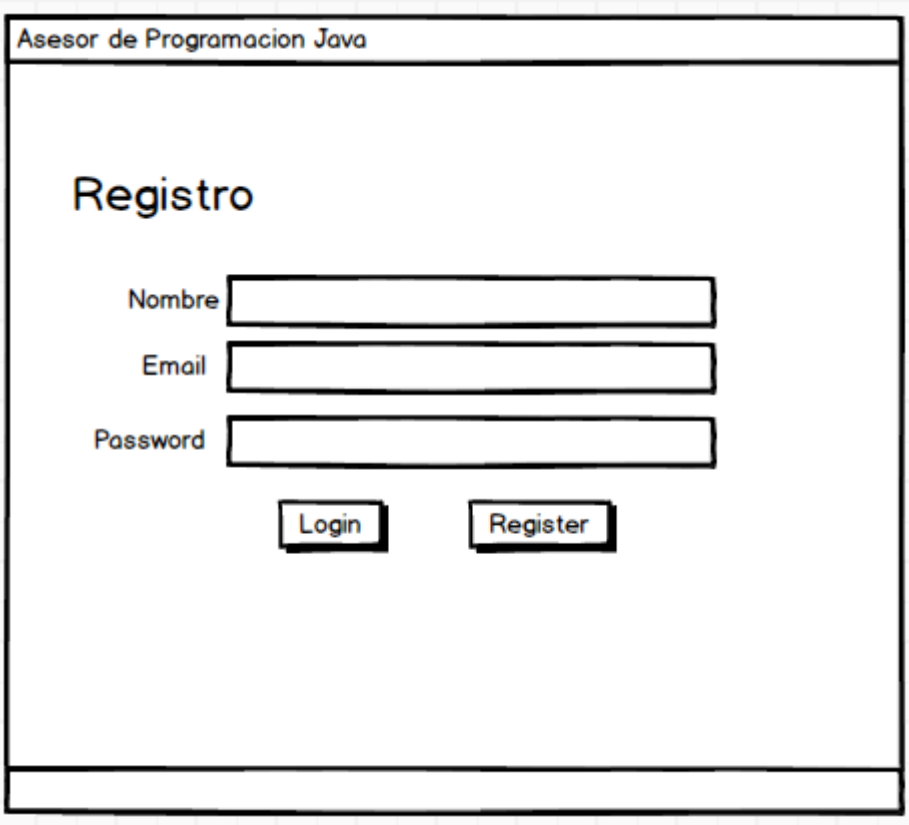

Fig. 12: Plantilla de Registro

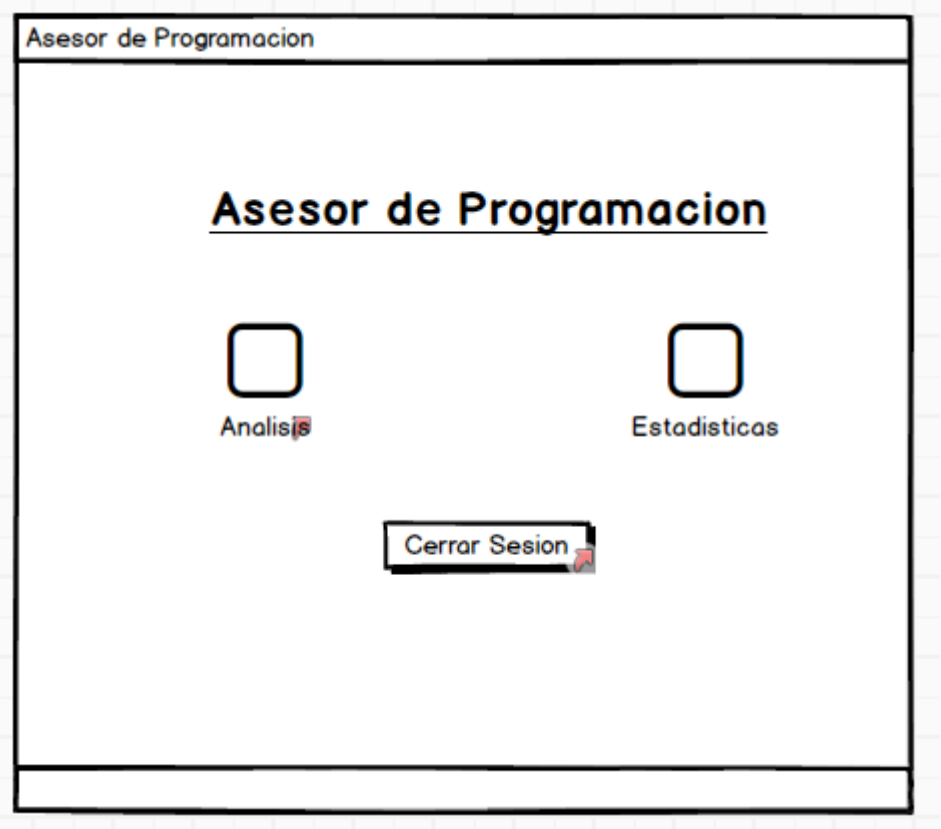

Fig. 13: Plantilla de la pantalla principal

Fig. 14: Plantilla para la elección del análisis.

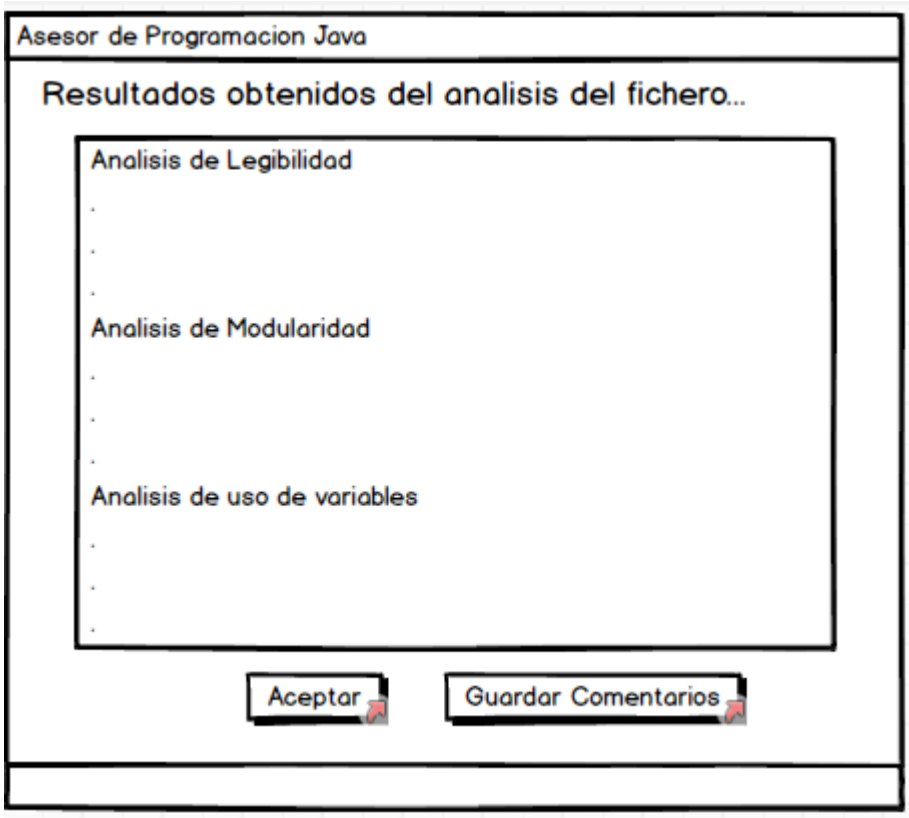

Fig. 15: Plantilla para mostrar los resultados obtenidos.

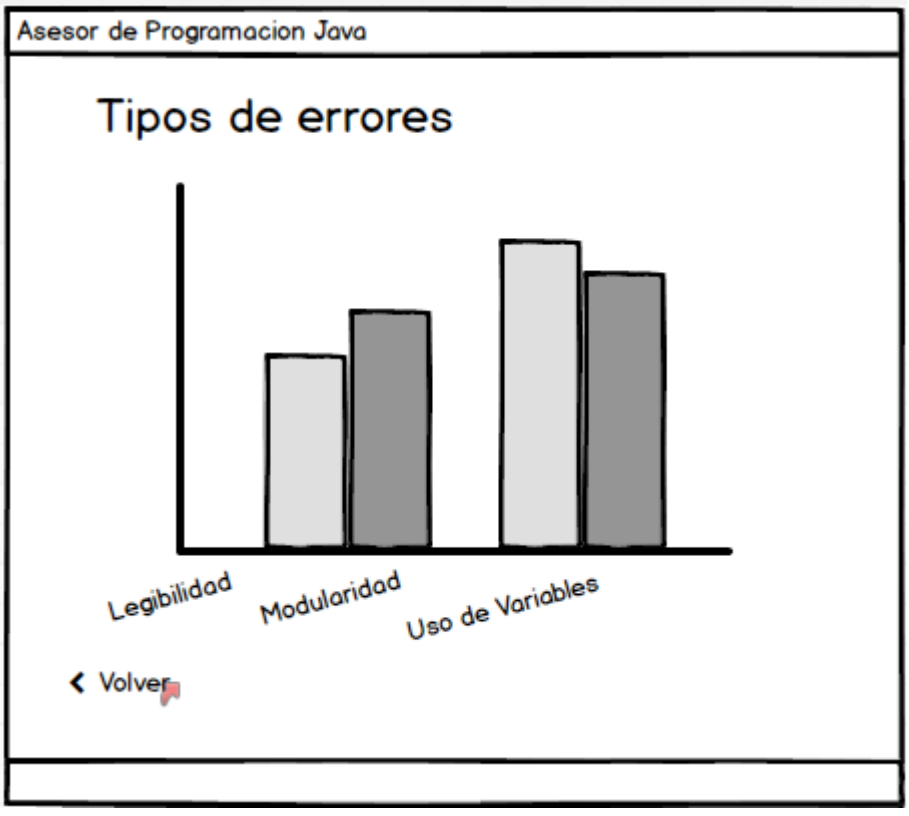

Fig. 16: Plantilla de gráficos.

4.4. Diseño del controlador.

El controlador de la aplicación es un Servlet que se encarga de recoger todas las peticiones que llegan desde la vista, realizar una funcionalidad específica y devolver un resultado de lo que ha sucedido al usuario.

Las peticiones que procesa el controlador pueden ser peticiones Get y peticiones Post, las peticiones Get se utilizan para consultar datos como por ejemplo visualizar las gráficas; y las peticiones Post se utilizan para insertar o modificar datos, por ejemplo, el registro de usuarios.

Las funcionalidades que realiza el controlador son las descritas en los requisitos funcionales y se han llevado a cabo a través de clases y métodos en lenguajes de programación Java. Estas funcionalidades se encargar de realizar todos los pasos necesarios para obtener un resultado que se enviará a la vista para poder ser visualizado.

4.5. Diseño del modelo.

Como se ha mencionado anteriormente el sistema estará formado por una BBDD formada por dos tablas que en un principio no estarán relacionadas entre sí.

La primera tabla Usuarios, contiene los registros de los usuarios de la aplicación, permitiendo el acceso a la utilización de esta. La tabla está formada por los campos de:

- Id usuario: identificador auto incrementable asignado automáticamente a cada nuevo usuario. Tipo de dato: Int.
- Nombre: nombre del usuario de la aplicación. Tipo de dato: Varchar2.
- Email: email del usuario registrado. Tipo de dato: Varchar2.
- Password: contraseña elegida por el usuario. Tipo de dato Varchar2.

Fig. 17: Tabla de usuarios.

La segunda tabla fallos, contiene los registros de la cantidad de fallos de cada tipo que se producen al realizar los diferentes tipos de análisis. Esta tabla se utiliza para poder

mostrar los datos estadísticos de la aplicación. La tabla está formada por los siguientes campos:

- Id Resultado: identificador auto incrementable asignado automáticamente a cada nuevo tipo de análisis. Tipo de dato: Int.
- Tipo de análisis: nombre del tipo de fallo registrado (Legibilidad, Modularidad, UsoVariable). Tipo de dato: Varchar2.
- Total: cantidad de fallos registrados para todos los análisis realizados en la aplicación. Tipo de dato: Number.

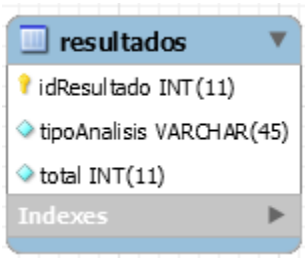

Fig. 18: Tabla de resultados.

5. IMPLEMENTACION DEL SISTEMA.

En este apartado se muestra la estructura y los tipos de ficheros por los que está formado la versión final del proyecto. Toda la implementación ha sido llevada a cabo utilizando las herramientas mencionadas del apartado 4.2.

La interfaz del sistema (Vista) está formado por ficheros de tipo JSP y CSS, estos son los encargados de mostrar los datos correctos en cada una de las diferentes pantallas y añadir los estilos personalizados respectivamente. Los ficheros JSP están codificados en lenguaje HTML5 y JAVA y los CSS en lenguaje CSS3. Estos ficheros podemos encontrarlos en el proyecto en la siguiente ubicación:

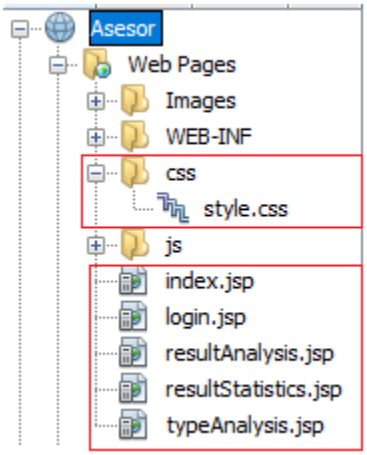

Fig. 19: Ubicación de las vistas en el proyecto.

La funcionalidad del sistema recae sobre los ficheros de tipo JS y JAVA. Los ficheros JS están codificados en el lenguaje de programación de JavaScript y son los encargados de dar la funcionalidad al sistema a nivel de interfaz (no forman parte del controlador). Los ficheros JAVA están codificados en el lenguaje java y son los encargados de dar la funcionalidad a nivel de controlador e implementar los requisitos funcionales. Estos ficheros podemos encontrarlos en el proyecto en la siguiente ubicación:

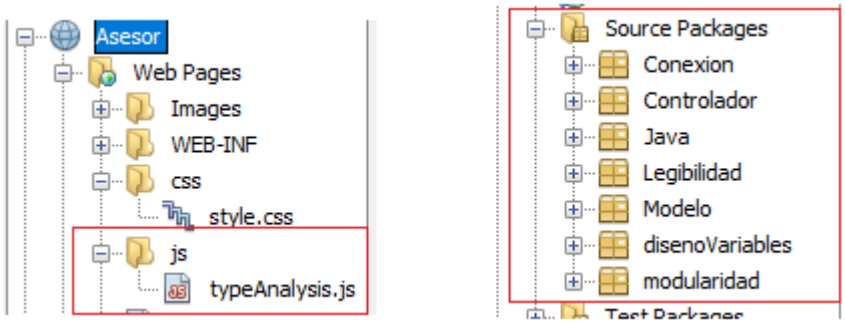

Fig. 20: Ubicación de clases de funcionalidad en el proyecto.

El lugar donde se almacenan los datos no se encuentra dentro del proyecto, si no que nos conectamos a ella a través del controlador, utilizando la librería externa de MySQL Connectors. Esta parte está programada en el lenguaje MySQL y a continuación se adjunta el Script de creación con el que obtener su estructura:

```
CREATE TABLE `usuarios` (
  `idusuarios` INT NOT NULL AUTO_INCREMENT,
  `email` VARCHAR(45) NOT NULL,
  `password` VARCHAR(45) NOT NULL,
  `nombre` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`idusuarios`),
  UNIQUE INDEX `email_UNIQUE` (`email` ASC));
```

Fig. 21: Script de creación de tabla de usuarios.

```
CREATE TABLE `resultados` (
  `idResultado` INT NOT NULL AUTO_INCREMENT,
  `tipoAnalisis` VARCHAR(45) NOT NULL,
  `total` INT NOT NULL,
  PRIMARY KEY (`idResultado`),
  UNIQUE INDEX `email_UNIQUE` (`tipoAnalisis` ASC));
```

Fig. 22: Script de creación de tabla de resultados.

6. EVALUACION DE RESULTADOS

En este apartado se va a comentar acerca del plan seguido para evaluar los resultados obtenidos al finalizar el desarrollo. Al tratarse de un proyecto software, para la evaluación del sistema se ha utilizado un sistema de pruebas que permite verificar el cumplimiento de requisitos especificados [12]. Las pruebas realizadas al sistema son las siguientes:

- Pruebas unitarias: pruebas realizadas a cada uno de los métodos de las clases controlando cual es la salida que se produce y si coincide con la esperada. [13]
- Pruebas de integración: una vez realizadas las pruebas unitarias y obtener un resultado exitoso, se realizan las pruebas de integración que constan de ir ensamblando cada uno de los componentes (clases) en el sistema final (proyecto).[14]
- Pruebas de validación: una vez montado todo el sistema, se procede a realizar las pruebas de validación o aceptación que constan de probar la funcionalidad completa del sistema y comprobar si cumple con los requisitos especificados. [15]

Para facilitar la visualización y el entendimiento de las pruebas se hará uso de las siguientes plantillas para las pruebas unitarias y de validación respectivamente:

Tabla 41: Plantilla de pruebas unitarias.

PU-X	
Nombre	Nombre de la prueba unitaria.
Descripción	Breve descripción de la prueba realizada.
Clase	Clase afectada.
Método	Método de la clase afectado.
Validación	Resultado de la validación entre el resultado obtenido y el resultado esperado.

Tabla 42: Plantilla de pruebas de validación.

PV-X	
Nombre	Nombre de la prueba de validación.
Descripción	Breve descripción de la prueba realizada.
Requisitos afectados.	Requisitos que se validan si la pruebas es correcta.
Validación	Resultado de validación de la prueba.

6.1. Pruebas unitarias.

En este apartado se van a realizar las pruebas unitarias a cada uno de los métodos del proyecto, se considera que una prueba ha sido validad cuando el resultado esperado coincide con el resultado obtenido.

Tabla 43: Prueba unitaria 1

PU-1	
Nombre	Análisis de indexación.
Descripción	Comprobar que el método es capaz de controlar la indexación de un texto y almacenar en una lista las líneas del código que no están correctamente indexadas.
Clase	Legibilidad
Método	analizarIndexacion()
Validación	Validado

Tabla 44: Prueba unitaria 2

PU-2	
Nombre	Análisis de comentarios.
Descripción	Comprobar que el método es capaz de identificar las subrutinas de un programa y comprobar que las precede un comentario, si no es así, se almacena en una lista la subrutina correspondiente.
Clase	Legibilidad
Método	analizarComentarios()
Validación	Validado

Tabla 45: Prueba unitaria 3

PU-3	
Nombre	Análisis de saltos incondicionales.
Descripción	Comprobar que el programa es capaz de detectar un salto incondicional y almacenarlo en una lista el numero de la linea.
Clase	Legibilidad
Método	analizarSaltos()
Validación	Validado

Tabla 46: Prueba unitaria 4

PU-4	
Nombre	Análisis de nombres de variables y funciones.
Descripción	Comprobar que el programa detecta cuales son las variables, constantes, procedimientos y funciones del programa y si siguen un formato determinado a lo largo de todo el programa(CamelCase), almacenando en una lista el número de línea que no cumple dicho formato.
Clase	Legibilidad
Método	analizarNombres()
Validación	Validado

Tabla 47: Prueba unitaria 5

PU-5	
Nombre	Comprobar variable no inicializada.
Descripción	Comprobar que el programa detecta si una variable ha sido inicializada en algún punto del código, si no es así, el programa almacena en una lista el número de línea donde se definen dichas variables.
Clase	UsoVariables
Método	comprobarAsignacionValor()
Validación	Validado

Tabla 48: Prueba unitaria 6

PU-6	
Nombre	Múltiple asignación a una variable.
Descripción	Comprobar que el programa detecta cuando una variable ha sido asignada mas de una vez y almacena en una lista si esta variable no ha sido utilizada entre asignaciones.
Clase	UsoVariables
Método	usoValorEntreAsignacion()
Validación	Validado

Tabla 49: Prueba unitaria 7

PU-7	
Nombre	Uso de variable.
Descripción	Comprobar que el programa detecta cuando una variable no ha sido utilizada y almacena en una lista el número de línea donde se define la variable.
Clase	UsoVariables
Método	seUtilizaVariable()
Validación	Validado

Tabla 50: Prueba unitaria 8

PU-8	
Nombre	Acceso a variable global por visibilidad.
Descripción	Comprobar que el programa detecta que se accede a una variable global del código por visibilidad y almacena en una lista el numero de la línea donde se produce.
Clase	UsoVariables
Método	noUsarVariablesGlobales()
Validación	Validado

Tabla 51: Prueba unitaria 9

PU-9	
Nombre	Modificar parámetros control de bucles.
Descripción	Comprobar que el programa detecta si se esta modificando las variables de control de los bucles, si es así, almacena en una lista la línea donde se modifica dicha variable.
Clase	Bucles
Método	analizarBucle()
Validación	Validado

Tabla 52: Prueba unitaria 10

PU-10	
Nombre	Asignación de valor a variable de retorno.
Descripción	Comprobar que el programa detecta si la variable de retorno de una función tiene valor o no, si no tienen valor almacena en una lista la línea donde se declara la función.
Clase	Funciones
Método	analizarCuerpoFuncion()
Validación	Validado

Tabla 53: Prueba unitaria 11

PU-11	
Nombre	Modularidad de subprograma.
Descripción	Comprobar que el programa es capaz de controlar el tamaño de un subprograma y almacenar en una lista si este excede el tamaño definido como máximo (20 líneas)
Clase	Programa
Método	contarLineasSubprograma()
Validación	Validado

Tabla 54: Prueba unitaria 12

PU-12	
Nombre	Encontrar valor de retorno de funciones y procedimientos.
Descripción	Comprobar que el programa es capaz de detectar si una función o un procedimiento tienen valores de retorno. Si la función no tiene valor de retorno se almacenará en una lista la línea correspondiente a la definición de la función. Por lo contrario, si un procedimiento tiene valor de retorno, se almacenara en una lista la línea donde se encuentra la definición del procedimiento.
Clase	Programa
Método	encontrarValorRetorno()
Validación	Validado

Tabla 55: Prueba unitaria 13

PU-13	
Nombre	Modificación de argumentos
Descripción	Comprobar que el programa es capaz de controlar si se está modificando un argumento de una función o un procedimiento, y almacena en una lista el número de la línea donde se produce esta acción.
Clase	Programa
Método	modificaArgumentos()
Validación	Validado

Tabla 56: Prueba unitaria 14

PU-14	
Nombre	Detección de expresión de impresión por pantalla.
Descripción	Comprobar que un programa que calculo datos (contiene argumentos) no debe realizar una impresión por pantalla, si no es así, se almacena en una lista el numero de la línea donde se realiza dicha impresión.
Clase	Programa
Método	encontrarExpresionImprimir()
Validación	Validado

Tabla 57: Prueba unitaria 15

PU-15	
Nombre	Detección de expresión de dato pedido por pantalla.
Descripción	Comprobar que el programa detecta cuando se pide un dato por pantalla y que le precede un mensaje previo, si no es así, almacena en una lista el número de la línea donde se pide el dato.
Clase	Programa
Método	encontrarExpresionTeclado()
Validación	Validado

Tabla 58: Prueba unitaria 16

PU-16	
Nombre	Localizar mensaje sacado por pantalla.
Descripción	Comprobar que el programa detecta cuando se realiza una impresión por pantalla y que le precede un mensaje previo, si no es así, almacena en una lista el número de la línea donde se realiza la impresión.
Clase	Programa
Método	encontrarMensajePrevio()
Validación	Validado

6.2. Prueba de integración.

Una vez realizadas todas las pruebas unitarias y comprobadas que funcionan correctamente, se realizan las pruebas de integración, que consta de ir unificando todas las partes en un proyecto en común.

El orden seguido para la integración es el siguiente:

- Crear un proyecto para almacenar cada uno de los componentes del sistema.
- Añadir las clases de funcionalidad JAVA al proyecto e insertar las dependencias correspondientes.
- Añadir las interfaces del sistema.
- Redireccionar todas las peticiones de las interfaces al controlador para que éstas se puedan comunicar correctamente con las clases de funcionalidad.
- Conectar el controlador a la base de datos utilizando los drivers necesarios y las credenciales ésta.
- Verificar que se pueden realizar correctamente todas las acciones disponibles desde la web.

6.3. Pruebas de validación.

Cuando se termina el proceso de integración, se procede con las pruebas de validación que consistirán en ir probando todas las funcionalidades de la aplicación

desde la interfaz y comprobando que los resultados obtenidos son los que se esperaban.

Tabla 59: Prueba de validación 1

PV-1	
Nombre	Registro de usuarios.
Descripción	Se realiza el registro de varios usuarios desde la aplicación.
Resultado obtenido	Los usuarios creados desde la aplicación se insertan correctamente en el base de datos.
Requisitos afectados.	RF-1
Validación	Validado

Tabla 60: Prueba de validación 2

PV-2	
Nombre	Ingreso de usuarios.
Descripción	Se realiza el login con usuarios registrados y no registrados.
Resultado obtenido	Solamente si el usuario esta registrado nos permite acceder al menú principal de la aplicación.
Requisitos afectados.	RF-2
Validación	Validado

Tabla 61: Prueba de validación 3

PV-3	
Nombre	Realizar análisis de código.
Descripción	Se realiza una prueba de cada uno de los tipos (Legibilidad, Modularidad y Uso de variables) y una de todos juntos de un código con errores controlados
Resultado obtenido	La aplicación detecta todos los errores del código y los muestra al usuario.
Requisitos afectados.	RF-3, RF-6, , RF-7, , RF-8, , RF-9, , RF-10, , RF-11, RF-12, RF-13, RF-14, RF-15, RF-16, RF-17, RF-18, RF-19, RF-20, RF-21, RF-22, RF-23, RF-24, RF-25, RF-26, RF-27, RF-28, RF-29, RF-30, RF-31 y RF-32
Validación	Validado

Tabla 62: Prueba de validación 4

PV-4	
Nombre	Descargar fichero de comentarios.
Descripción	Cuando se realiza un análisis, se descargar el fichero de comentarios recopilados por la aplicación del fichero de código a analizar.
Resultado obtenido	La aplicación recoge todos los comentarios en el fichero descargado.
Requisitos afectados.	RF-33
Validación	Validado

Tabla 63: Prueba de validación 5

PV-5	
Nombre	Visualizar grafico de errores
Descripción	Se accede a la vista para visualizar los gráficos estadísticos de errores.
Resultado obtenido	La aplicación pinta correctamente los gráficos con los valores almacenados en la base de datos.
Requisitos afectados.	RF-4 y RF -5
Validación	Validado

7. PLANIFICACIÓN Y PRESUPUESTO

En este apartado se hace referencia a la planificación que se ha llevado a cabo para realizar el proyecto y el presupuesto que supondría llevar a cabo un proyecto de estas características.

7.1. Planificación de tareas.

A continuación, se adjunta una tabla donde se definen las tareas realizadas con su duración en días y el diagrama de Gantt que muestra las tareas a lo largo del proyecto.

Tabla 64: Tareas realizadas en el proyecto.

Id	Nombre de la tarea	Duración	Inicio	Fin	Predecesoras
1	Definición de requisitos.	1d	17/01/2018	17/01/2018	
2	Estudio de viabilidad de los requisitos.	2d	18/01/2018	19/01/2018	1
3	Análisis final de requisitos.	1d	22/01/2018	22/01/2018	2
4	Elección del modelo a seguir en el desarrollo.	2d	23/01/2018	24/01/2018	3
5	Implementación de funcionalidad del sistema.	30d	25/01/2018	07/03/2018	4
6	Pruebas unitarias del sistema	7d	08/03/2018	16/03/2018	5
7	Maqueta de la interfaz	1d	08/03/2018	08/03/2018	3; 5
8	Diseño de la interfaz.	10d	09/03/2018	22/03/2018	7
9	Implementación de funcionalidad de la interfaz	7d	23/03/2018	02/04/2018	3; 8

10	Integración de vista con controlador	3d	03/04/2018	05/04/2018	5; 9
11	Pruebas de integración de vista con controlador	5d	03/04/2018	09/04/2018	5; 9
12	Diseño de la base de datos	1d	19/03/2018	19/03/2018	3; 6
13	Implementación de la base de datos	2d	20/03/2018	21/03/2018	12
14	Integración de la base de datos con controlador	3d	20/03/2018	22/03/2018	6; 12
15	Pruebas de integración de base de datos con controlador	5d	10/04/2018	16/04/2018	14; 11
16	Pruebas de validación del sistema	10d	10/04/2018	23/04/2018	10; 14; 11
17	Elección de estructura del contenido de la memoria	1d	24/04/2018	24/04/2018	16
18	Análisis, diseño e implementación del sistema	17d	25/04/2018	17/05/2018	17
19	Introducción, conclusiones y líneas futuras del sistema	8d	18/05/2018	29/05/2018	18
20	Presupuesto del proyecto	2d	30/05/2018	31/05/2018	19
21	Análisis del estado del arte	5d	01/06/2018	07/06/2018	20

22	Planificación del proyecto	2d	08/06/2018	11/06/2018	21
23	Revisión final de documentación	2d	12/06/2018	13/06/2018	22

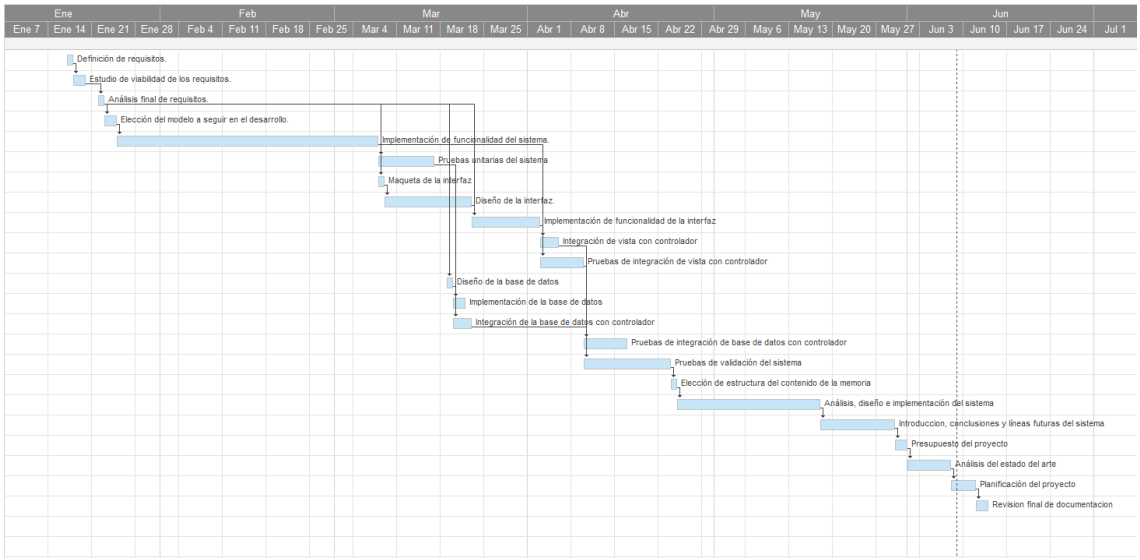

Fig. 23: Diagrama de Gantt

En el diagrama podemos se observar que la tarea que más tiempo ha llevado ha sido la de la implementación de la parte funcional del sistema, que a su vez es la más crítica ya que muchas otras dependen de esta. Por tanto, la mayor parte del esfuerzo de este proyecto fue enfocado a realizar una correcta implementación que permitiese avanzar con el resto de las tareas.

7.2. Presupuesto

En este apartado se hace mención de los gastos que han supuesto llevar a cabo un proyecto de estas características.

Para una mejor identificación del tipo de gasto, se agruparán en gastos de software, de hardware, otros o mano de obra, dependiendo de su origen.

Gastos de Software

Tabla 65: Gastos de software

Nombre	Unidades	Meses	Precio unitario	Gasto
NetBeans	1	6	0,00	0,00
Visual Studio Code	1	6	0,00	0,00
MySQL Workbench	1	6	0,00	0,00
Office 365 Hogar	1	6	99,00	99,00*
LucidChar	1	6	0,00	0,00
Balsamiq	1	6	9,00	54,00
SmartSheet	1	6	13,00	78,00
Total				**231,00**

(*) Gasto final es igual al gasto unitario porque es una licencia por producto, no mensual.

Gastos de Hardware

El precio unitario de los activos materiales será el precio de amortización por mes calculado de manera lineal mediante la siguiente formula [16]:

$$Amotizacion\ mensual = \frac{Valor\ de\ adquisición}{Meses\ vida\ útil}$$

Ecuación 1: Amortización mensual de activos materiales

Se toma como vida útil del activo la proporcionada por el distribuidor a la hora de adquirir el activo.

El valor de adquisición de pc de desarrollo asciende a la cantidad de 1218€, los meses de vida útil son 48(4 años) y aplicando la fórmula de amortización se obtiene un resultado de 25,38.

El valor de adquisición del pc de pruebas y visualización asciende a la cantidad de 860€, los meses de vida útil son 48 (4 años) y aplicando la formula con estos datos se obtiene un resultado de 17,9.

Se decide obviar el cálculo de amortización del pendrive y se pone una cantidad irrelevante de 1€ por mes.

Tabla 66: Gastos de hardware

Nombres	Unidades	Meses	Precio unitario	Precio Total
Pc de desarrollo.	1	6	20,3	152,28
Pc de pruebas de visualización y revisión.	1	6	17,9	107,4
Pendrive	1	6	1,00	6,00
Total				**265,68**

Otros Gastos

Nombre	Unidades	Meses	Precio unitario	Precio Total
Luz	-	6	70,00	420,00
Internet	-	6	50,00	300,00
Lugar de trabajo	-	6	300,00	1800,00
Total				**2520**

Mano de obra

Tomando como referencia el salario bruto anual de un desarrollador de 20,740€ [17] tenemos que obtener el salario mensual para poder incluirlo en el gasto en el proyecto con la siguiente fórmula:

$$Salario\ mensual = \frac{Salario\ Anual\ medio}{12}$$

Ecuación 2: Salario mensual del Ingeniero

Aplicando la formula no sale un salario mensual de 1728,33

Nombre	Unidades	Meses	Precio unitario	Precio Total
Ingeniero informático Junior	-	6	1728,33	10369,98
Total				**10369,98**

Para obtener el coste final del proyecto, solamente tendremos que sumar los costes parciales generados a lo largo del proyecto, lo que asciende a una cantidad de 13.368,66 €.

$$Coste\,Total = Coste\,software + Coste\,Hardware + Otros\,Gasots + Mano\,de\,Obre$$

Ecuación 3: Coste total proyecto

8. CONCLUSIONES Y LINEAS FUTURAS.

8.1. Objetivos cumplidos.

Una vez realizada la herramienta de “Asesor de programación” nos toca recapitular y ver si hemos conseguido los objetivos que nos marcamos al principio del proyecto y podemos decir a ciencia cierta que todos los objetivos de funcionalidad, usabilidad, seguridad, escalabilidad y modularidad han sido cumplidos con éxito.

En lo relacionado con la funcionalidad de la herramienta ha cumplido con creces todos los objetivos de controlar la funcionalidad especificada en los requisitos, lo que permitirá a cualquier usuario novel poder realizar un análisis previo de su código y corregir los errores antes de entregarlo para una evaluación o simplemente por interés propio de mejorar la calidad de su código.

En lo relacionado con la usabilidad, se ha diseñado una interfaz sencilla e intuitiva para que el usuario no sienta ningún rechazo hacia la herramienta y se encuentre cómodo usándola. Para facilitar aún más el uso de la herramienta se adjuntará como anexo un manual de usuario para el aprendizaje de uso de la herramienta y el flujo de ejecución de esta.

En lo relacionado con la seguridad, todos los datos de los usuarios estarán regidos por la ley orgánica de protección de datos y todas las transacciones de la aplicación se hacen de una manera segura.

Con respecto a la escalabilidad y modularidad, se ha comprobado que la aplicación esta compuestas por varios módulos, no solamente modelo-vista-controlador, si no que dentro del propio código se ha organizado en paquetes para facilitar su mantenimiento o añadir alguna nueva funcionalidad al sistema.

Como objetivo personal podría decir que la realización de este proyecto me ha ayudado a profundizar aun mas sobre mis conocimientos tanto de fronEnd como de backEnd. También se mencionaba al comienzo de este proyecto que nos podría servir para aprender algún tipo de tecnología desconocida y así ha sido, aprendí el manejo del framework de Bootstrap para implementar interfaces de mayor calidad y la herramienta de SmartSheet para realizar diagramas de Gantt.

Como experiencia personal realizando este proyecto me gustaría destacar, que a medida que la herramienta adquiría nueva funcionalidad, los resultados obtenidos me llegaban incluso a sorprender, cuando realizaba pruebas de funcionalidad introduciendo ficheros con errores controlados, sin poner ninguna atención en los otros

tipos de errores, la aplicación detectaba los otros tipos de errores y los notificaba, persónamele quede muy sorprendido.

8.2. Líneas futuras de trabajo.

A lo largo del proceso de creación del proyecto han ido surgiendo pequeñas ideas con lo que poder mejorar con nueva funcionalidad esta herramienta y que, bajo mi punto de vista, resultarían muy interesante si algún día se llegaran a implementar.

A continuación, se listan las posibles mejoras:

- Realizar una herramienta más compleja, ya no solo enfocada a usuarios noveles, si no para usuarios especialistas, que pueda analizar el rendimiento del programa y pueda asesor de como poder mejorar dicho rendimiento.
- Escalar la herramienta a más lenguajes de programación, no solo el lenguaje de programación de Java, lo que permitiría a usuarios expandir sus conocimientos en los distintos lenguajes.
- Permitir al usuario realizar la corrección de su propio código desde la herramienta.
- Mostrar ejemplos al usuario del error cometido y como corregirlo.
- Mostrar estadísticas más detalladas aplicando filtros de análisis, de usuarios, de años o comparaciones con errores en otros lenguajes de programación.
- Añadir un sistema de puntuación por fallos, que permita evaluar el código en función del tipo de errores y la ponderación proporcionada.

Se podrían seguir listando mejoras a realizar en la herramienta que expandiría su funcionalidad y aumentaría el número de usuarios a los que va dirigido, pero creo que con estas mejoras se obtendría una herramienta más ambiciosa.

BIBLIOGRAFÍA

[1] T. Fernández , «Así es ya el nuevo empleo tecnológico,» *Expansion,* 07 05 2018.

[2] A. Alonso, «adrianalonso,» 2 12 2017. [En línea]. Available: https://adrianalonso.es/formacion/los-principios-de-desarrollo-solid-en-5-videos/.

[3] M. Brocca, «josefachin,» 31 01 2018. [En línea]. Available: https://josefacchin.com/ley-proteccion-de-datos/.

[4] iso25000, «iso25000,» 2018. [En línea]. Available: http://iso25000.com/index.php/normas-iso-25000/iso-25010?limit=3&limitstart=0.

[5] E. Munguía, «enrique7mc,» 11 05 2016. [En línea]. Available: http://www.enrique7mc.com/2016/05/los-10-errores-mas-frecuentes-en-programacion/.

[6] M. Rancel, «aprendeaprogramar,» 2006. [En línea]. Available: https://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=340:tipos-de-errores-en-programacion-de-compilacion-o-ejecucion-gestionados-y-no-gestionados-cu00242a&catid=36&Itemid=60.

[7] M. Rancel, «aprenderaprogramar,» 2006. [En línea]. Available: https://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=332:prevencion-gestion-y-tipos-de-errores-errores-de-sintaxis-por-procesos-no-validos-y-logicos-cu00241a&catid=36&Itemid=60.

[8] C. Pes, «carlospes,» 2006. [En línea]. Available: http://www.carlospes.com/curso_de_ingenieria_del_software/04_04_tipos_de_errores.php.

[9] D. Ramos, R. Noriega, J. R. Láinez y A. Durango, «Casos de uso,» de *Curso de Ingeniería de Software: 2ª Edición*, Createspace Independent, 2017.

[10] D. Ramos, R. Noriega, J. R. Láinez y A. Durango, «Requisitos funcionales y no funcionales,» de *Curso de Ingeniería de Software: 2ª Edición*, Createspace

Independent, 2017.

[11] M. Á. Álvarez, «Desarrolloweb,» 02 01 14. [En línea]. Available: https://desarrolloweb.com/articulos/que-es-mvc.html.

[12] M. Cillero, «manuel.cillero,» [En línea]. Available: https://manuel.cillero.es/doc/metrica-3/tecnicas/pruebas/.

[13] M. Cillero, «manuel.cillero,» [En línea]. Available: https://manuel.cillero.es/doc/metrica-3/tecnicas/pruebas/unitarias/.

[14] M. Cillero, «manuel.cillero,» [En línea]. Available: https://manuel.cillero.es/doc/metrica-3/tecnicas/pruebas/integracion/.

[15] M. Cillero, «manuel.cillero,» [En línea]. Available: https://manuel.cillero.es/doc/metrica-3/tecnicas/pruebas/aceptacion/.

[16] J. Valencia, «Economipedia,» 2015. [En línea]. Available: http://economipedia.com/definiciones/amortizacion-contable-lineal.html.

[17] indeed, «indeed,» 12 6 2018. [En línea]. Available: https://www.indeed.es/salaries/Desarrollador/a-junior-Salaries.

ANEXO A: GLOSARIO

CamelCase	Estilo de escritura aplicado a palabras compuestas, donde la primera letra de cada palabra comienza por mayúscula.
Casos de uso	Descripción de los pasos que hay que dar en un sistema para realizar alguna acción. En la ingeniería del software describen la interacción entre el usuario y el sistema.
CSS3	lenguaje de diseño gráfico, utilizado para la visualización de interfaces web desarrolladas en HTML.
Diagramas Gantt	herramienta grafica que permite visualizar la duración de una tarea a lo largo del tiempo.
Diagramas UML	son la combinación de diversos elementos gráficos para formar diagramas explicativos. El lenguaje de UML cuenta con sus propias normas para la relación de sus elementos.
HTML5	lenguaje de programación en su quinta versión utilizado para crear páginas web.
Java	lenguaje de programación rápido, seguro y fiable. Este lenguaje se encuentra presente en muchas aplicaciones y sitios web.
JavaScript	lenguaje de programación web que permite mejorar la interfaz de usuario y la construcción de web dinámicas.
JSP	(Java Servlet Pages) software que permite la creación de páginas dinámicas combinado lenguaje de programación java y HTML.
Requisitos del sistema	necesidad documentada de un sistema sobre su funcionalidad o su forma.
Servlet	clase en lenguaje de programación Java, que puede escuchar y responder peticiones web y ampliar la funcionalidad de un servidor.
SQL	lenguaje de programación específico para realizar operaciones en base de datos.

ANEXO B: MANUAL DE USUARIO

A continuación, se listan y se describen los pasos a seguir para realizar un uso correcto de la aplicación de “Asesor de programación”

Paso 0: Registro en la aplicación.

Para poder hacer uso de la aplicación lo primero que tenemos es que registrarnos introduciendo todos los datos que aparecen el formulario. Se debe tener en cuenta que no se permite que dos usuarios tengan el mismo email.

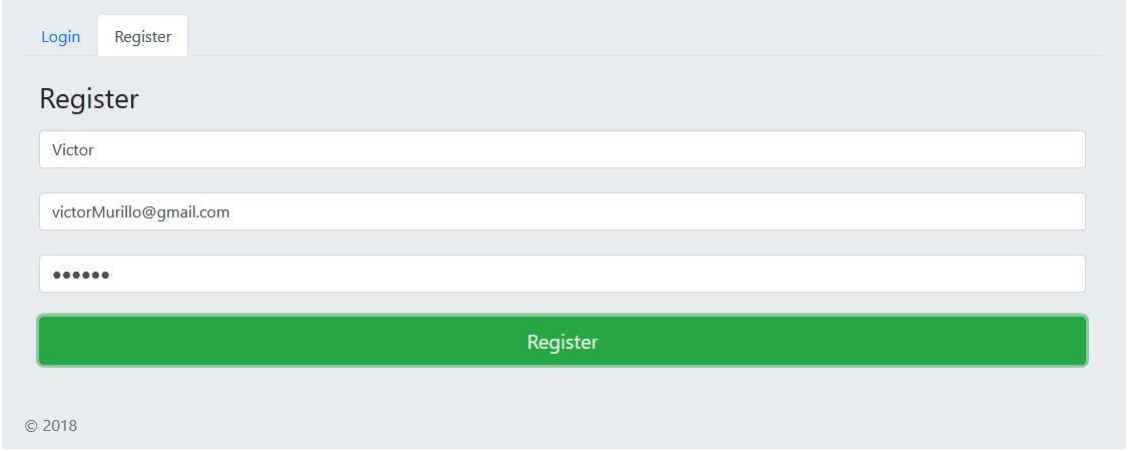

Fig. 24: Registro en aplicación

Paso 1: Ingreso en aplicación

Una vez registrados en la aplicación, se ingresarán las credenciales correctas en la pantalla de Login para poder acceder a la vista principal del sistema.

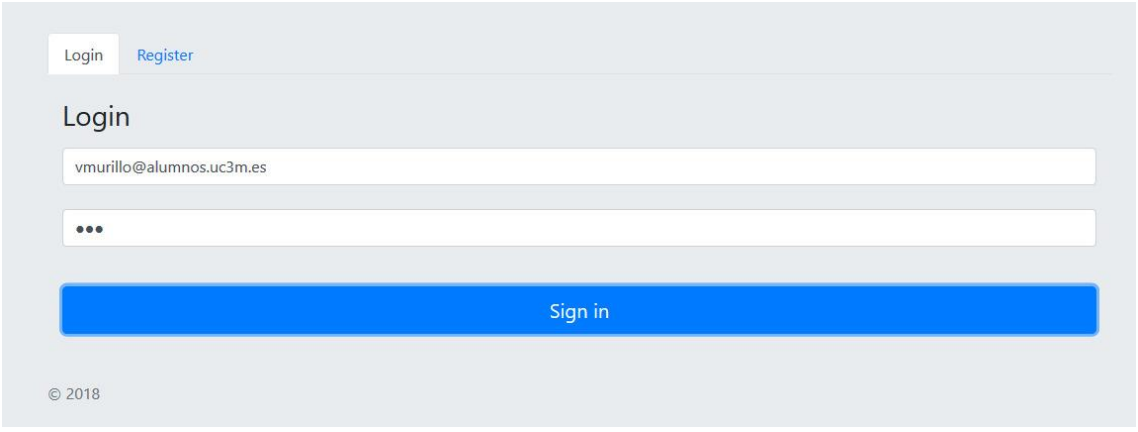

Fig. 25: Ingresar en aplicación

Paso 2: Vista principal de la aplicación

En el menú principal de la aplicación se podrá realizar una de las dos acciones disponibles Visualizar Gráficos o Realizar Análisis.

Fig. 26: Vista principal de aplicación

Paso 3.1: Realizar análisis

En esta pantalla nos permitirá seleccionar el fichero de código a analizar y seleccionar el tipo de análisis a realizar. Hay que tener en cuenta que, si no seleccionamos un fichero no nos dejara seleccionar ningún tipo de análisis, y si no seleccionamos ningún análisis no podremos realizar la acción.

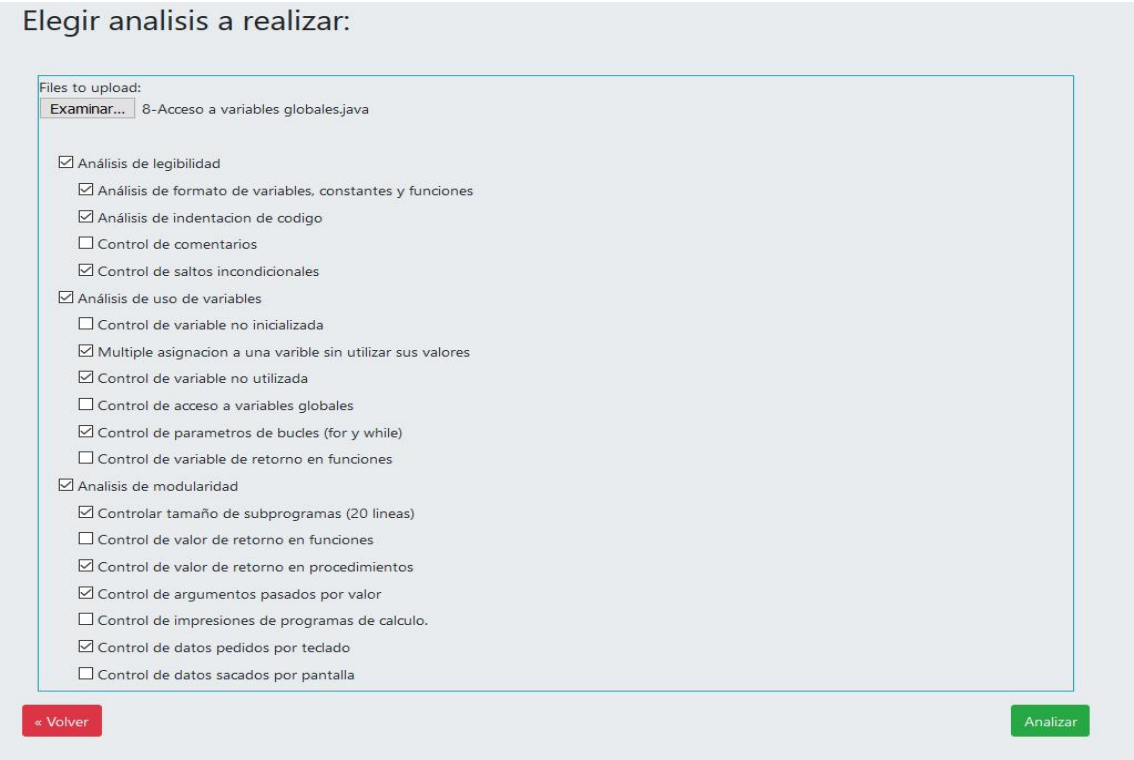

Fig. 27: Elección de fichero y análisis a realizar

Paso 3.1.1: Visualización de resultados

En esta pantalla se muestran los resultados obtenidos tras realizar el o los análisis seleccionados, también permite la descarga de los comentarios encontrados en el fichero.

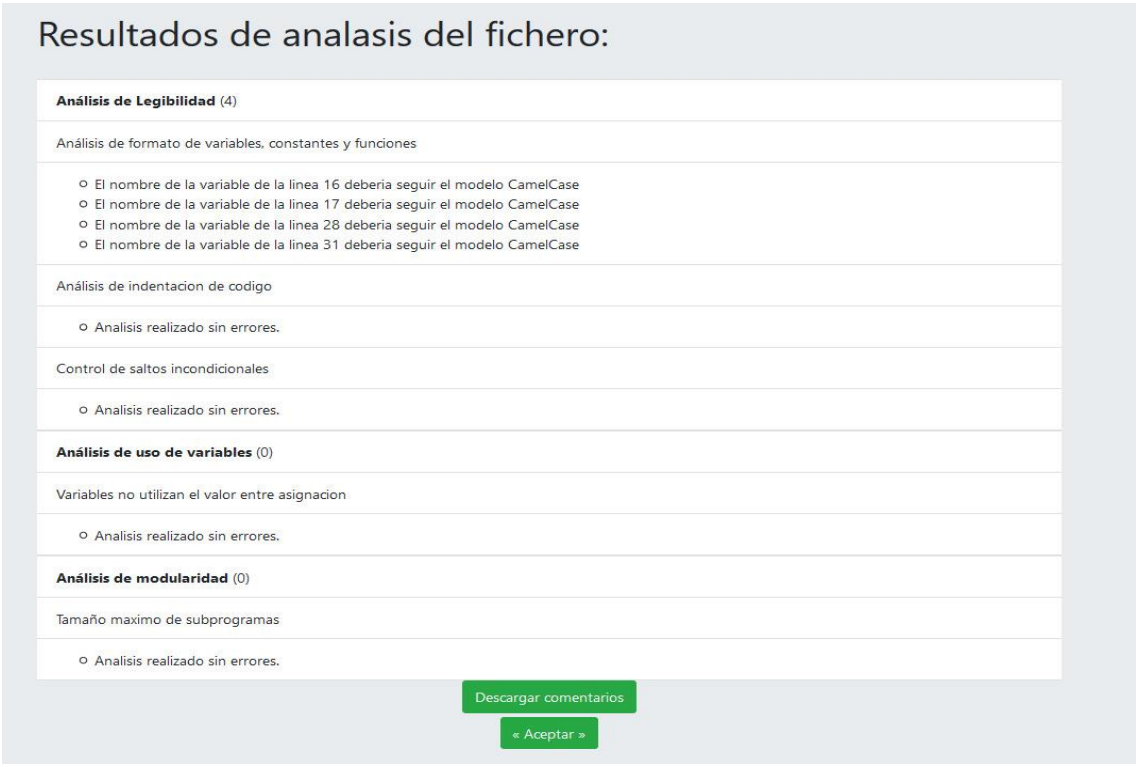

Fig. 28: Visualización de resultados

Paso 3.2: Visualizar gráficos

En esta pantalla se muestran un diagrama de barras con los fallos acumulados por todos los usuarios que utilizan la aplicación.

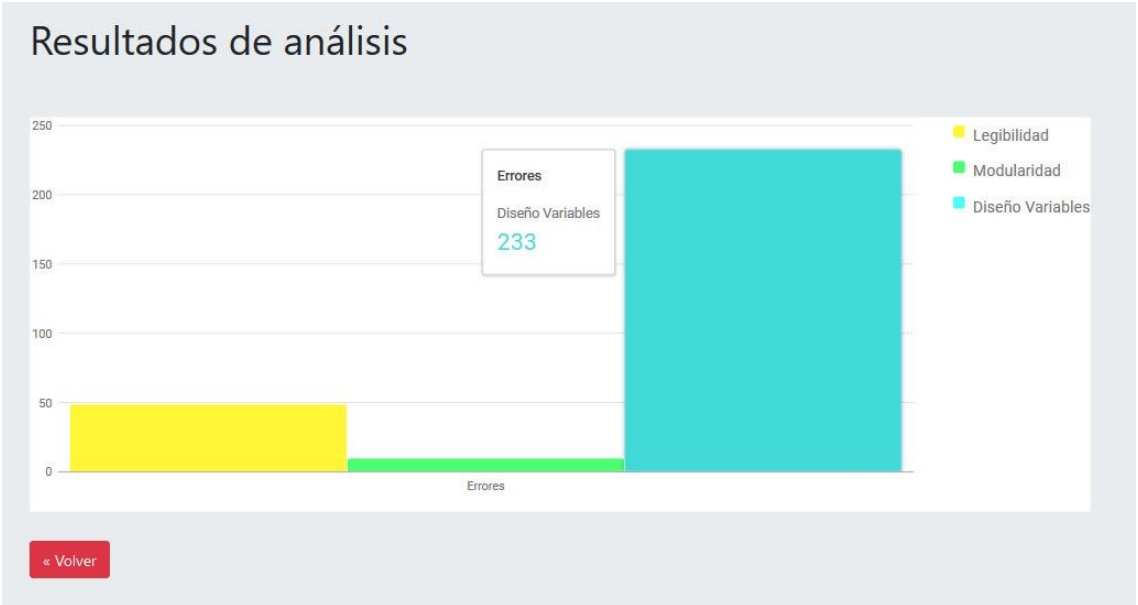

Fig. 29: Visualización de gráficos de resultados.

ENGLISH VERSION

Introduction

The reason why it has been decided to carry out this project is to be able to help people who are starting in programming, helping them to understand the errors that are occurring in the codes they implement and that are able to correct these errors for themselves.

The main objective of the project is to reduce the number of errors in programming codes in the Java language by obtaining higher quality codes.

As a secondary objective it is intended that the students of the universities use this tool to reduce the suspense index in subjects of this style.

The fact of making a web application, we must take into account the aspects, the organic data protection law in force in Spain and the software product quality standard ISO_2500, in order to avoid any quality conflict and create a software of quality.

State of the art

The current situation in which we find ourselves according to an analysis carried out by the University of Kent, which analyzed millions of codes of new students of Java programming, it was deduced which were the most common errors.

The following is a summary of these errors:

- -Incorrect use of parentheses, curly brackets, brackets or quotation marks.
- Calls to methods with the wrong number of arguments.
- Do not return value in functions.
- Confuse the assignment operator = with the comparison operator ==.

Once all the errors are controlled, quality codes are produced, although this is not an easy task since it takes a lot of practice and experience to detect them.

To delve more deeply into how to solve errors, they will be divided into two large sets, compilation errors and execution errors.

Compile errors are what prevent the execution of the code. They can be divided into two subtypes:

- Syntax errors: errors caused by not writing an instruction correctly.
- Errors of processes not valid in compilation: errors produced when calling a method that does not exist or its construction is not correct.

On the other hand, we have the execution errors, these errors are those that are not detected by the compiler and allow the execution of the program, but not in a correct way. They can be divided into the following subtypes:

- Errors of processes not valid in execution: these errors occur when an impossible action is attempted.
- Logical infinite loop errors: these errors occur when the control parameters of a loop are not managed properly.
- Logic of incorrect result: these errors are those committed by the programmer, you want the program to follow a logic, but it has not been implemented correctly.

The project is implemented in the NetBeans environment, an environment that helps us to easily detect compilation errors and which I personally recommend to new people in this field.

As most programming environments allow us to detect compilation errors, an application focused on detecting execution errors and creating readable, modular programs and a correct variable design will be made.

The application that is made has the following characteristics:

- Web Application
- User Management
- Diversity of analysis to be performed
- Help to make code documentation.
- Error statistics of all users.

With the design of this tool, it is intended that the novice user is able to correct for himself his own failures and at the same time learn, so that in the future he can avoid them and fulfill all the objectives set.

Analysis of the system

The system is designed for code analysis of new users who are taking their first steps in Java programming. With the use of the system, this type of user will be able to easily locate the errors of the code and be able to reach the objective of producing a quality code.

Before implementing the system, the needs of the product must be identified, for which purpose identification is made using the use case technique using UML diagrams, which represent the interaction of a user with the system. The identification of use cases makes it easier for us to obtain requirements that the system wants to meet. You can see the cases of use in section 3.2 of the document.

In this section we will describe the requirements that the system must meet. We will classify the requirements into two types: functional and non-functional. The functional requirements refer to the functionality of the system and the non-functional to how the functionalities are going to be performed.

You can visualize the functional requirements in section 3.3.1 of the document and the non-functional requirements in section 3.3.2.

To verify that all the functional requirements are related to any use case, a traceability matrix is made that allows to see this relationship and analyze which will be the most critical requirements depending on the number of cases that are affected.

System design.

This section specifies the architecture that the system will have, the technological environment where it will be developed and the design that is followed to make each of the different layers of architecture.

The architecture chosen for the system is model-view-controller (MVC). The MVC is characterized by the separation of the application into three different layers to divide the responsibility of the application.

- Model: is the layer that is related to the data, this layer will be responsible for storing, deleting or making changes to the data.
- View: is the layer related to the interface, it will be the layer responsible for the user to interact with the system, making requests and visualizing the results.

- Controller: is the layer that connects the view with the model, is the layer where all the logic of the application is done in order to make sense of the requests made.

The hardware development environment used is a desktop PC and the software tool, Netbeans, you can see the characteristics of the hardware and the rest of the tools used to perform the project in section 4.2 of the document.

In section 4.3, you will find the design mockup of the application interface.

Implementation of the system

In this section the structure and the types of files by which the final version of the project is formed are shown.

The system interface (Vista) consists of files of type JSP and CSS, these are responsible for displaying the correct data in each of the different screens and add the custom styles respectively.

The functionality of the system falls on the files of type JS and JAVA. The JS files are encoded in the JavaScript programming language and are responsible for giving the system functionality at the interface level. The JAVA files are encoded in the java language and are responsible for giving the functionality at the controller level and implementing the functional requirements.

The data of the application is stored in a relational database, programmed in SQL, you can find the creation script in section 5 of this document.

Evaluation of results

In this section we will comment on the plan followed to evaluate the results obtained at the end of the development. Being a software project, for the evaluation of the system has been used a test system that allows to verify the fulfillment of specified requirements. The tests performed on the system are:

- Unit tests
- Integration testing
- Validation tests.

All tests performed on the system are collected in section 6 of this document.

Planning

In this section reference is made to the planning that has been carried out to carry out the project and the budget that would involve carrying out a project of these characteristics.

In this section we attach a table with the tasks performed and their duration in days in the project, a Gantt chart is also attached that allows us to visualize how these tasks have been developed over time.

On the other hand, mention is made of the budget necessary to carry out a project of these characteristics, dividing the expenses according to their origin and then adding each of these to obtain the total cost of the project.

Conclusions and future lines.

Once the "Programming Advisor" tool has been made, we have to recap and see if we have achieved the objectives that we set ourselves at the beginning of the project and we can say with certainty that all the objectives of functionality, usability, security, scalability and modularity have been fulfilled successfully.

Regarding the functionality of the tool, it has fulfilled all the objectives of controlling the functionality specified in the requirements, which will allow any new user to perform a prior analysis of their code and correct errors before submitting it for an evaluation. or simply for self-interest to improve the quality of your code.

Regarding usability, a simple and intuitive interface has been designed so that the user does not feel any rejection towards the tool and is comfortable using it. To facilitate the use of the tool even more, a user manual will be attached as an annex to learn how to use the tool and how to use it.

In relation to security, all user data will be governed by the organic data protection law and all transactions of the application are made in a secure manner.

Regarding scalability and modularity, it has been proven that the application is composed of several modules, not only model-view-controller, but that within the code itself it has been organized in packages to facilitate its maintenance or to add some new functionality to the system.

As a personal objective I could say that the realization of this project has helped me to deepen even more about my knowledge of fronEnd and backEnd. It was also mentioned at the beginning of this project that we could learn some kind of unknown technology and that's how it was, I learned how to use the Bootstrap framework to implement higher quality interfaces and the SmartSheet tool to create Gantt diagrams.

As a personal experience doing this project I would like to emphasize that as the tool acquired new functionality, the results obtained even surprised me, when I performed functionality tests introducing files with controlled errors, without paying any attention to the other types of errors, the application detected the other types of errors and notified them, persuade him to be very surprised.

Throughout the process of creating the project have been emerging small ideas with which to improve this tool with new functionality and, in my view, would be very interesting if someday they were implemented.

Below are the possible improvements:

- Make a more complex tool, no longer only focused on new users, but for specialized users, who can analyze the performance of the program and can advise on how to improve such performance.
- -To scale the tool to more programming languages, not only the Java programming language, which would allow users to expand their knowledge in different languages.
- -Allow the user to correct their own code from the tool.
- -Show examples to the user of the error and how to correct it.
- -Show more detailed statistics by applying analysis filters, users, years or comparisons with errors in other programming languages.
- -Add a system of punctuation by failures, that allows to evaluate the code based on the type of errors and the weighting provided.

You could continue listing improvements to be made in the tool that would expand its functionality and increase the number of users to which it is addressed, but I believe that with these improvements a more ambitious tool would be obtained.